UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Desenvolvimento de um processo automático de gestão de vulnerabilidades de ciber segurança em ambientes de grande dimensão

Fábio Guimarães Fernandes

Mestrado em Engenharia Informática
Especialização em Arquitectura, Sistemas e Redes de Computadores

Trabalho de projeto orientado por:
Prof. Doutor António Casimiro Ferreira da Costa
Eng. José António dos Santos Alegria

2019

Agradecimentos

No meu percurso académico este foi sem dúvida o projeto mais desafiante e ambicioso em que estive envolvido. Por este estar inserido na área de segurança informática pela qual tenho especial interesse tornou o desafio mais entusiasmante. Pelas razões mencionadas anteriormente agradeço ao Eng. José Alegria pela oportunidade de realizar este projeto e pela orientação dada ao longo do tempo.

Para a equipa de segurança Web onde estive durante a primeira parte do projeto um agradecimento por todo o apoio dado e conhecimento partilhado, em especial, para o Tiago Mendo. Agradeço também à equipa do SOC pela forma como me receberam na segunda parte do projeto e por todo o apoio prestado, especificamente ao Jorge Silva pela partilha de conhecimento e orientação na fase final da concretização. Para o Ricardo Ramalho e os elementos da equipa de desenvolvimento, a qual acabei por integrar, um agradecimento pelo apoio prestado na integração com as plataformas, pelo suporte durante algumas fases mais complicadas do desenvolvimento e pelo apoio motivacional na fase final.

Ao professor António Casimiro um especial agradecimento pela orientação, paciência e perseverança demonstradas pelo que sem ele a conclusão não teria sido possível. Um agradecimento também para os meus colegas de faculdade pelo espírito de entreajuda demonstrados ao longo do curso.

Em último mas não menos importante um agradecimento à minha família e aos meus amigos, em especial aos meus pais e irmã, pelo apoio prestado ao longo da realização do curso.

À minha família e amigos.

Resumo

A persistente ameaça de ataques por parte de criminosos informáticos na Internet faz com que as informações confidenciais de grandes empresas estejam em constante risco. A perda ou exposição destas pode causar grandes prejuízos às empresas e, com isto em mente, as empresas empregam mecanismos típicos de segurança como *firewalls* e sistemas de deteção de intrusões.

Todavia, estes mecanismos são apenas eficientes contra ataques originados no exterior da rede empresarial por ligações não autorizadas e não necessárias para o funcionamento dos sistemas que suportam os serviços das empresas. Nas ligações não bloqueadas podem surgir ataques para os quais os sistemas não estão protegidos. A crescente adoção de dispositivos móveis faz com que os colaboradores, que são utilizadores autorizados, sejam vistos como possíveis ameaças de ataques internos na rede.

Visto isto, torna-se essencial proteger a um nível individual os ativos críticos que contêm informação confidencial. De forma a proteger os ativos é necessário descobrir as vulnerabilidades que estes contêm para ser possível mitiga-las futuramente.

Este projeto tem como objetivo o estudo e concretização de uma solução de descoberta e análise de vulnerabilidades em ativos críticos controlados na vertente de segurança pela Direção de Cyber Security e Privacidade (DCY) da Portugal Telecom (Altice Portugal). Esta solução é responsável por agendar *scans*, orquestrar os dois *scanners* de vulnerabilidades, *OpenVAS* e *Nexpose*, e enviar os dados para as plataformas usadas pela DCY, *ArcSight* e Hidra.

A solução desenvolvida tem como objetivo o auxílio na gestão de vulnerabilidades feita pela DCY aos ativos críticos que esta tem de proteger.

Palavras-chave: Segurança, Vulnerabilidades, Gestão de Vulnerabilidades, Análise de Risco

Abstract

The persistent threat of attacks perpetrated by cybercriminals in the Internet puts confidential information belonging to large organizations in constant danger. The loss or exposure of this information may cause big losses to the organizations affected. With this in mind, organizations deploy typical defence mechanisms like firewalls and intrusion detection systems (IDS).

But these mechanisms are only effective at stopping attacks that originate outside the enterprise network by unauthorized users.

The growing adoption of mobile devices increase, the possibility of employees, which are authorized users, being seen as likely threats to the inside network. In light of this it becomes essential to protect individual critical assets that contain confidential information. In order to protect the assets it is necessary to discover their vulnerabilities so these in the future can be mitigated. This project aims to study and implement a solution to discover and analyse vulnerabilities in critical assets controlled in the security area by the Directorate of Cyber Security and Privacy (DCY) of Portugal Telecom (Altice Portugal). This solution is responsible for scheduling scans, orchestrating the two vulnerability scanners, OpenVAS and Nexpose, and send the data to the platforms used by DCY, ArcSight and Hidra.

The solution aims to aid in the process of vulnerability management by the DCY to the critical assets that they must protect.

Keywords: Security, Vulnerabilities, Vulnerability Management, Risk Analysis

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente uma infraestrutura informática é um requisito fundamental para o sucesso de qualquer empresa, dado que esta projeta a imagem da empresa para o exterior e facilita a gestão dos processos de negócio, quer externa quer internamente.

Então será seguro dizer que a infraestrutura informática de uma empresa está intrinsecamente ligada à sua reputação.

Tendo isto em conta, torna-se essencial para qualquer empresa investir na segurança dos seus ativos informáticos.

No contexto de empresas de grande dimensão, o processo de proteger devidamente os seus sistemas informáticos é uma tarefa difícil, pois estas contêm um grande número de atores que interagem com estes sistemas, assim como uma quantidade elevada de ativos que dependem uns dos outros para manter o bom funcionamento do negócio.

Tipicamente a prioridade das empresas é a proteção a ataques originários do exterior, levando a que empreguem mecanismos que diminuam a superfície de ataque dos seus sistemas em relação ao exterior e impeçam intrusões que comprometam a segurança e saúde destes. Um dos exemplos mais conhecidos destes mecanismos é a *firewall* que tem como objetivo controlar o tráfego de e para a empresa, permitindo o que está autorizado e negando o restante. Esta quando bem configurada diminui bastante a superfície de ataque a uma rede interna. Outro mecanismo muito utilizado é o IDS/IPS (*Intrusion Detection/Prevention System*) que, já na rede interna, deteta e previne ataques reportando-os ao administrador da rede, permitindo que este possa proteger melhor os sistemas.

Frequentemente as empresas depositam confiança excessiva nestes mecanismos, esquecendo-se do facto que estes estão mais direcionados para proteger de ataques de utilizadores externos e não autorizados.

Ao delinear-se um plano de segurança de uma empresa é imprudente e ingénuo depositar confiança total nos elementos internos, em especial empresas que possuem um grande número de colaboradores.

Com a grande proliferação do uso de dispositivos computacionais móveis (portáteis, *smartphones*, *wearables*, etc.) os colaboradores introduzem na rede interna das empresas um grande número de dispositivos que, fora do horário de trabalho, se encontram ligados a outras redes mais inseguras onde, possivelmente, estão expostos a ataques por parte de adversários que podem infetar estes dispositivos. Isto traz um risco acrescido pois esses dispositivos podem aceder a credenciais dos colaboradores e podem espalhar *malware* a outros dispositivos da rede.

Esta projeto está inserido no âmbito da cadeira de Projeto de Engenharia Informática do 2º ano curricular do Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa.

O projeto foi desenvolvido autonomamente na Portugal Telecom (Altice Portugal), mais especificamente na Direção de Cyber Security e Privacidade (DCY).

Este projeto tem como missão alertar para a importância da segurança numa perspetiva individual dos ativos pertencentes a um sistema informático numa empresa, de forma a dotá-los de uma maior resistência a ataques externos ou internos à empresa.

A falta de segurança de um ativo está ligada a vulnerabilidades que existem no *software* que corre neste, em especial no *software* que está exposto à rede. A correção destas vulnerabilidades é então de extrema importância para a melhoria de segurança de ativos na rede.

1.1 Motivação

Para uma empresa é crucial manter níveis altos de segurança nos seus sistemas informáticos. Uma intrusão bem sucedida pode comprometer uma ou mais propriedades de segurança. No caso de um atacante obter acesso privilegiado a um sistema, facilmente viola a **confidencialidade**, **integridade** e **disponibilidade** dos dados da empresa, ou, ainda sem acesso privilegiado, um ataque de negação de serviço pode desabilitar a **disponibilidade** dos sistemas de uma empresa de forma a diminuir ou anular a sua capacidade de fornecer serviços. Todas estas situações podem levar uma empresa a ter enormes prejuízos que podem levar à sua falência, como foi o caso da autoridade de certificação DigiNotar[7][37].

Tendo isto em consideração a maioria das empresas implanta mecanismos tradicionais de defesa, como *firewalls* ou IDS, que estão vocacionados para proteger a rede da empresa de ataques de utilizadores externos não autorizados. Caso um adversário consiga comprometer um dispositivo de um colaborador, pode utilizar este como porta de entrada para atacar internamente a rede da empresa. Caso os ativos internos da empresa tenham um nível de segurança baixo um adversário pode facilmente roubar ou adulterar dados confidenciais da empresa e dos seus clientes,

desligar sistemas e destruir dados, infetar máquinas com *malware* para possibilitar novos ataques, etc.

A existência de bases de dados online públicas com milhares de vulnerabilidades e o lançamento diário de várias vulnerabilidades novas nestas, com detalhes técnicos sobre como as explorar, apresenta-se como um elemento que auxilia na proteção de ativos, especialmente se existirem correções para as vulnerabilidades, pois permite que uma equipa de segurança descubra e corrija estas vulnerabilidades. Mas ao mesmo tempo torna-se uma desvantagem visto que um atacante pode usufruir dessa informação para atacar ativos que ainda não se encontrem protegidos.

Numa empresa de grande dimensão a tarefa de proteger individualmente todos os seus ativos é bastante complexa. O número elevado de ativos e a sua heterogeneidade em termos de hardware e software, bem como a elevada quantidade de colaboradores que interagem com estes, são dois fatores que tornam difícil o controlo necessário para proteger os sistemas de uma empresa.

Tendo em conta os fatores descritos em cima torna-se óbvio que, para defender um sistema complexo de grandes dimensões, é necessário um mecanismo automático que auxilie na descoberta e gestão de vulnerabilidades.

1.2 Objectivos

O objetivo principal deste projeto consiste no estudo, desenho, desenvolvimento e avaliação de uma solução para monitorizar e medir o nível de vulnerabilidades de um painel de ativos críticos em relação a um *baseline* predefinido por essa mesma solução. Entende-se por *baseline* o estado inicial em termos de vulnerabilidades de um conjunto de ativos.

A solução a desenvolver deve providenciar o seguinte conjunto de funcionalidades:

- Orquestração da execução, de forma periódica, de um conjunto predefinido de ferramentas designadas *vulnerability scanners*, sendo estas responsáveis por pesquisar vulnerabilidades num conjunto fornecido de ativos.

- Normalização das informações de vulnerabilidades, de forma a existir uma representação única que possa ser utilizada para análise

- Envio destas informações para o SIEM *ArcSight* de forma a alertar as equipas operacionais da DCY, responsáveis pela segurança, caso exista uma situação urgente relacionada com vulnerabilidades num ativo.

- Transmissão destas informações para a plataforma Hidra (*High performance Infrastructure for Data Research and Analysis based on Elasticsearch*) que é

uma base de dados de alto desempenho e disponibilidade focada para investigação na área de segurança na DCY. Esta plataforma engloba várias tecnologias: *RabbitMQ*, *Elasticsearch*, *Logstash*, *Kibana*.

Previamente foram definidos os *vulnerability scanners* a ser utilizados, *OpenVAS* e *Nexpose*. O *Nexpose* é uma ferramenta que precisa de uma licença com um custo monetário associado sendo que o *OpenVAS* é *open-source* e gratuito. Este tipo de ferramentas cria um grande número de pedidos para testar vulnerabilidades e isso faz com que estas sobrecarreguem a rede e os ativos. Estes fatores levam a que um dos objetivos deste trabalho fosse a definição de uma estratégia para a execução das duas ferramentas de forma a minimizar o custo monetário da licença e o impacto na rede.

A estratégia passa pela realização de um *scan* primário com o OpenVAS e consoante as características do ativo deverá ser feito um *scan* secundário com o *Nexpose*.

Também é necessário um estudo aprofundado das ferramentas de forma a afinar as suas configurações com o objetivo de minimizar o impacto, sacrificando o mínimo de precisão. Outro objetivo é a definição de uma estrutura de métricas de vulnerabilidades para possibilitar a medição de desvios face a um *baseline* pré-definido pela solução.

1.3 Tarefas realizadas

Seguidamente é feita uma apresentação das tarefas realizadas no projeto.

- Levantamento das matérias relevantes na área de vulnerabilidades de maneira a auxiliar o desenvolvimento do projeto e escrita desta dissertação, incluindo trabalhos similares que possam guiar este projeto.
- Estudo de tecnologias e ferramentas usadas na área de vulnerabilidades, especialmente aquelas que foram ser usadas neste projeto. Foi feito um estudo mais aprofundado sobre as ferramentas cruciais à realização do projeto, o *OpenVAS* e *Nexpose*.
- Pesquisa e levantamento do funcionamento dos sistemas utilizados pela DCY que interagem com a solução a desenvolvida, nomeadamente o sistema de gestão de eventos de segurança *ArcSight* e base de dados de análise de segurança Hidra, analisando de forma especial o esquema de armazenamento a ser utilizado para vulnerabilidades.
- Planeamento e desenho da arquitetura da solução desenvolvida e os seus módulos.
  - Sistema de agendamento de *scanners*.

  - Esquema de decisão de aplicação de múltiplos *scanners*.
  - Módulos de normalização de informação de vulnerabilidades para cada *scanner*.
  - Módulos de envio da informação para o *ArcSight* e Hidra..

- Concretização da solução tendo em conta o desenho da sua arquitetura e realização de testes preliminares para permitir uma análise prévia às informações geradas por este.
- Definição da estrutura de métricas a ser usada, de forma a permitir, no Hidra, a medição de desvios face aos níveis de base dos ativos.
- Teste da solução num conjunto de ativos predefinidos e análise dos resultados obtidos de forma a tirar conclusões sobre a validade e utilidade dos dados.
- Elaboração do documento da dissertação.

1.4 Organização do documento

Este documento é composto por 6 capítulos.

O primeiro capítulo (Cap. 1) é uma introdução ao contexto dos problemas de segurança em que se enquadra este projeto.

No capítulo 2 é feita uma exposição de conceitos teóricos e práticos de segurança relacionados com o projeto. Também são apresentadas ferramentas e standards empresariais que foram estudados para a realização do projeto. No fim do capítulo é feita uma exposição das plataformas da DCY que irão ser utilizadas no contexto do projeto.

O problema principal e os desafios do projeto são descritos de uma forma geral no capítulo 3. Também é apresentada a arquitetura inicial para a solução. Adicionalmente são expostos alguns desafios no ambiente em que a aplicação vai operar.

A concretização da aplicação, o seu funcionamento e a configuração de todos os elementos que interagem com esta são explicados no capítulo 4.

No capítulo de Avaliação (Cap. 5) é apresentado um conjunto de testes que se focam em testar as funcionalidades desenvolvidas, com o objetivo de cumprir os requisitos propostos para a aplicação.

O último capítulo (Cap. 6) descreve uma breve conclusão sobre o trabalho realizado, as mais-valias que a aplicação pode trazer no contexto da direção onde foi desenvolvida e alguns aspetos da aplicação que podem ser melhorados no futuro.

Capítulo 2

Contexto de trabalho

Neste capítulo é realizada uma exposição de conceitos teóricos e práticos de segurança que auxiliaram na compreensão dos objetivos do projeto. Também é feita uma exposição de ferramentas e plataformas que são parte integrante deste projeto.

Um ativo pode estar exposto a dois tipos de ataques, **ataques ativos**, onde um adversário envia diretamente um fluxo malicioso de dados de forma a explorar uma vulnerabilidade comprometendo a segurança da máquina, e **ataques passivos** onde o adversário escuta a rede de forma a aceder ou inferir informações relativas a comunicações realizadas [41]. Este projeto foca-se principalmente na defesa de ataques ativos, portanto é pertinente perceber em detalhe como estes são realizados.

2.1 Fases de um ataque - Modelo AVI

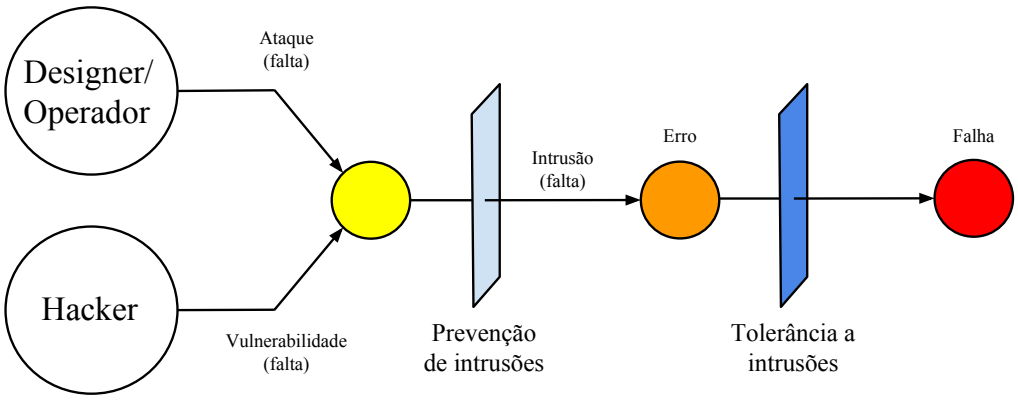

Figura 2.1: Modelo AVI

O Modelo AVI tem como propósito explicar as ações e condições necessárias para existência de uma intrusão.

Uma **intrusão** resulta de um **ataque** perpetrado por um adversário que explora com sucesso uma **vulnerabilidade** que o sistema possuía (Figura 2.1)[40]. Se o ataque for realizado com sucesso possibilita a violação das propriedades de segurança,

levando a que o adversário possa comprometer a confidencialidade, integridade e disponibilidade do sistema e/ou dos seus dados. Adicionalmente um atacante ainda pode instalar um *backdoor* para permitir futuras intrusões, mesmo que a vulnerabilidade explorada no ataque inicial seja corrigida.

Facilmente se percebe que a segurança de um sistema está fortemente relacionada com a presença de vulnerabilidades neste, o que leva a concluir que para proteger um sistema é necessário descobrir e corrigir as suas vulnerabilidades. O foco deste projeto encontra-se na primeira linha de defesa de um ataque, ou seja, na **prevenção de intrusões**, mais especificamente na descoberta de vulnerabilidades em sistemas.

De forma a descobrir e corrigir vulnerabilidades é necessário perceber melhor o que estas são.

2.2 Vulnerabilidades

Uma **vulnerabilidade** é um defeito ou uma fragilidade que pode ser explorada maliciosamente por um atacante de forma a comprometer a segurança do sistema [40][41][39].

Uma vulnerabilidade pode ser introduzida num sistema nas seguintes fases da sua existência[39]:

- **Desenho**: durante o desenho do software, por exemplo, na escolha de um mecanismo fraco de autenticação.
- **Concretização**: bugs introduzidos no código do sistema, por exemplo, falha na verificação correta de um *buffer* na memória.
- **Operação e Manutenção**: causado pelo ambiente em que o software corre ou pela sua configuração, por exemplo, contas por omissão ativas com credenciais conhecidas.

A severidade de uma vulnerabilidade está associada a vários fatores que condicionam o sucesso de um ataque. Sendo um dos principais a exequibilidade que está associada ao nível de acesso (remoto, local), complexidade técnica e necessidade de credenciais. Outro é o impacto que está associado aos atributos de segurança que uma vulnerabilidade pode comprometer (Confidencialidade, Integridade, Disponibilidade).

De seguida são descritos alguns dos mais comuns tipos de vulnerabilidades que afetam os sistemas atualmente[29].

Buffer Overflow: vulnerabilidade que permite que seja introduzido em memória um input maior do que aquele que foi reservado, possibilitando normalmente a um atacante escrever em zonas arbitrárias na memória do sistema. O atacante pode

utilizar esta vulnerabilidade para executar código arbitrário em modo privilegiado ou causar erros de forma a desativar o sistema. Para corrigir este tipo de vulnerabilidades é necessário limitar o tamanho do *input* a ser guardado em memória.

SQL Injection: permite que sejam executados comandos SQL arbitrários na base de dados através de um *input* malicioso especialmente criado, onde são injetadas *queries* SQL. Permite consultar, adicionar, alterar e remover dados de uma base de dados. A solução passa por parametrizar as *queries* SQL.

Command Injection: possibilita a execução de comandos arbitrários no sistema operativo do sistema alvo através de uma aplicação, levando a que um adversário possa usar esses comandos para revelar informações do sistema, escalar privilégios, desativar o sistema, etc. Para impedir um ataque devemos validar o *input* da aplicação.

Security Misconfiguration: vulnerabilidade que assenta numa configuração deficiente do sistema que permite que um adversário tenha acesso não autorizado ao sistema. Um exemplo é a manutenção de contas com credenciais por omissão ativas que podem ser facilmente usadas por adversários.

Cross-site scripting (XSS): tipo de vulnerabilidade em aplicações *Web* que permite que um adversário execute um *script* do lado do cliente, tipicamente *Javascript*, no *browser* de uma vítima. Esse *script* pode enviar dados da vítima provenientes de *cookies* ou do *browser* para o adversário.

Cross-Site Request Forgery (CSRF): esta vulnerabilidade permite que um atacante force um utilizador a executar ações numa aplicação *Web* onde este já se encontra autenticado. Por exemplo, um utilizador autentica-se na aplicação *Web* do banco e depois clica num *link* de um e-mail enviado pelo atacante, nesse *site* vai ser executado um *script* malicioso que interage com a aplicação *Web* do banco.

Como é expectável, o tipo de vulnerabilidades mais conhecidas e encontradas estão associadas a aplicações *Web*, visto estas aplicações tipicamente estarem expostas à Internet e, consequentemente, serem um alvo mais fácil de atacar.

No entanto, face à existência de uma equipa na DCY que gere um projeto que trata de vulnerabilidades *Web*, este projeto vai focar-se em vulnerabilidades de rede e sistemas. Uma vulnerabilidade num *software* que seja utilizado por um grande número de máquinas e serviços pode ter um grande impacto nas organizações, como é o caso do *Heartbleed* e o *Shellshock*, duas vulnerabilidades publicadas recentemente que tiveram um grande impacto na segurança das empresas.

Heartbleed: vulnerabilidade na biblioteca de criptografia *OpenSSL*[14]. A falta de validação de input na concretização da extensão *heartbeat* TLS faz com que exista uma vulnerabilidade *buffer over-read*, possibilitando um atacante ler a memória do sistema. Daí um atacante pode ler *passwords*, chaves privadas, dados de sessão, etc.

Shellshock: vulnerabilidade existente na *bash* que permite a um atacante exe-

cutar comandos arbitrários, sendo apenas possível se o atacante conseguir afetar o valor de variáveis ambiente do sistema com um conjunto de valores especiais seguido dos comandos que este quer executar[34]. Servidores *Web* com Linux normalmente usam a *bash* para processar comandos o que fez com que esta vulnerabilidade tivesse um grande impacto.

2.3 *Exploits*

Para provar que uma vulnerabilidade existe é necessário criar um programa que a explore. A esse programa dá-se o nome de ***exploit***[40] [41][39]. Um *exploit* que ainda não seja conhecido publicamente tem o nome de ***zero-day exploit***. O significado do nome deriva do facto de os responsáveis por um sistema ou *software* terem zero dias para mitigarem a vulnerabilidade[41][39].

Existem bases de dados públicas disponíveis na Internet com *exploits*, como por exemplo a *ExploitDB*[27] e o *Metasploit* [20]. Isto permite que uma equipa de segurança possa verificar com mais exatidão a possibilidade e a facilidade com que uma vulnerabilidade pode ser explorada, permitindo que se possam definir prioridades para a correção de vulnerabilidades. Ao mesmo tempo permite que um atacante com poucos conhecimentos técnicos consiga facilmente realizar ataques em sistemas que ainda não estejam protegidos.

2.4 Ciclo de vida de uma vulnerabilidade

Para compreender os desafios que trazem a gestão de vulnerabilidades de um sistema é essencial perceber o ciclo de vida de uma vulnerabilidade e as suas fases (Figura 2.2)[39][36][16]. A seguir são descritas, de uma forma generalizada, estas fases.

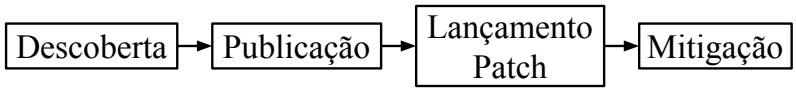

Figura 2.2: Fases do ciclo de vida de uma vulnerabilidade

Normalmente uma vulnerabilidade é **descoberta** por uma pessoa ou conjunto de pessoas, acidentalmente numa interação com uma aplicação ou intencionalmente através do estudo desta. Existem vários tipos de pessoas interessadas em descobrir vulnerabilidades:

- Equipas de segurança de *software* de aplicações, que se dedicam a tornar estas mais seguras, normalmente analisando o código fonte e eliminado *bugs*.

- Criminosos informáticos que descobrem vulnerabilidades para poderem executar ataques diretamente, espalhar *worms* para infetar o número máximo de dispositivos, ou vender *0-day exploits* para outros criminosos utilizarem.
- Empresas de segurança e entusiastas que descobrem vulnerabilidades pelo prestígio e/ou por uma compensação financeira que por vezes as empresas oferecem.

Quando uma equipa de segurança de *software* descobre uma vulnerabilidade, corrige-a e simultaneamente faz a **publicação** da vulnerabilidade e o lançamento do *patch*. No caso dos entusiastas e das empresas de seguranças, normalmente, as empresas que gerem as aplicações são avisadas sobre a vulnerabilidade com alguma antecedência antes da **publicação** por forma a dar tempo para a criação de um *patch*. No caso de uma vulnerabilidade ser utilizada por um criminoso informático esta só é publicada quando é detetada a sua exploração, por exemplo, quando após uma intrusão uma equipa de segurança analisa os sistemas e descobre as vulnerabilidades exploradas. Após o desenvolvimento do ***patch*** os utilizadores de um *software* são notificados e devem proceder o mais rápido possível à execução dos passos necessários à **mitigação** da vulnerabilidade. No caso dessa notificação falhar ou ser ignorada cabe às equipas de segurança verificar se estes *patches* são aplicados e as vulnerabilidades são efetivamente corrigidas.

2.5 Risco

No contexto de uma grande empresa existem um grande número de ativos para proteger. Tendo em conta que as equipas de segurança são recursos finitos é importante perceber como usar esses recursos da forma mais eficiente de forma a proteger os sistemas. Para isso é essencial definir prioridades em relação aos ativos a proteger. Uma estratégia possível é a proteção dos ativos mais críticos, visto estes serem essenciais para a realização das atividades fundamentais ao funcionamento de uma organização. Para perceber o nível de segurança de um ativo é necessário definir uma métrica que possibilite a comparação entre ativos. Essa métrica chama-se **risco**[40][41]. Normalmente é calculada em função do nível de **vulnerabilidade** do sistema, da **ameaça** a que este está exposto e do **impacto** do ataque. O nível de vulnerabilidade está associado ao número e à severidade das vulnerabilidades a que um sistema está exposto. O potencial de ataques que um sistema pode sofrer tem o nome de ameaça. O impacto está associado ao custo, que um ataque bem sucedido num sistema pode ter para uma organização. O impacto permite perceber quais os ativos mais críticos numa organização.

$$Risco = Vulnerabilidade \times Ameaça \times Impacto$$

2.6 *Common Vulnerabilities and Exposures* (CVE)

CVE é uma lista de referências de vulnerabilidades de segurança mantida pela *MITRE Corporation* que tem como objetivo ser uma base de dados comum a todo o sector de segurança informática[8]. Para isso providencia nomes comuns e identificadores únicos para problemas de segurança. Isto permite que se possam comparar e integrar diferentes ferramentas e sistemas de segurança que sejam compatíveis com CVE, facilitando o trabalho de gestão de vulnerabilidades na rede. A lista, *CVE List Master Copy*, está disponível publicamente no site para *download*. A publicação de um CVE segue um processo rigoroso onde é verificado se não existem referências de vulnerabilidades duplicadas e onde o conteúdo desta, nomeadamente a descrição, é rigorosamente definida tendo em conta as regras da lista.

Na lista CVE uma vulnerabilidade é normalmente identificada pelos seguintes campos:

- ***CVE-ID***: Este campo identifica univocamente uma vulnerabilidade na lista. Este é composto pelo prefixo CVE, seguido de quatro dígitos que representam o ano de publicação e acaba num número de tamanho arbitrário que identifica a vulnerabilidade no ano. Este número começa por ter 4 dígitos e aumenta consoante a necessidade associada ao número de vulnerabilidades publicadas nesse ano. Estes identificadores CVE são controlados pela *MITRE Corporation*, sendo esta a principal *CVE Numbering Authority* (CNA). A CNA principal distribui identificadores por blocos a outras CNA secundárias de forma a agilizar o processo da publicação. Estas CNA secundárias costumam ser as organizações que produzem o *software* onde os *bugs* são encontrados, sendo depois estas responsáveis por atribuir identificadores a investigadores de segurança que reportem vulnerabilidades no seu *software*.

- ***Description***: Campo que descreve brevemente a vulnerabilidade.

- ***Reference***: Este campo contém referências e informações adicionais sobre a vulnerabilidade, sendo normalmente *links*. Podem existir vários campos deste tipo repetidos para cada vulnerabilidade.

2.7 *Common Vulnerability Scoring System* (CVSS)

CVSS é um sistema padrão de avaliação de **severidade** de vulnerabilidades[6]. Define um conjunto de **métricas** que permite a definição de uma **pontuação** numérica de severidade de uma vulnerabilidade, e baseado nessa pontuação uma representação qualitativa. O CVSS permite que exista um sistema de pontuação comum a diferentes sistemas e ferramentas de vulnerabilidades. Isto permite criar uma visão global

da rede ao nível de gestão de vulnerabilidades, possibilitando que se possa analisar o estado de segurança global e individual de um conjunto de ativos, que por sua vez auxilia a criação de prioridades na mitigação de vulnerabilidades nesses ativos.

Este sistema de pontuação é usado em muitas ferramentas de segurança o que vai permitir que seja possível comparar e integrar resultados de ferramentas usadas neste trabalho e no futuro interagir com outras ferramentas que estão em produção na DCY.

A pontuação de uma vulnerabilidade é construída segundo um conjunto de 3 grupos de métricas apresentadas na Figura 2.3.

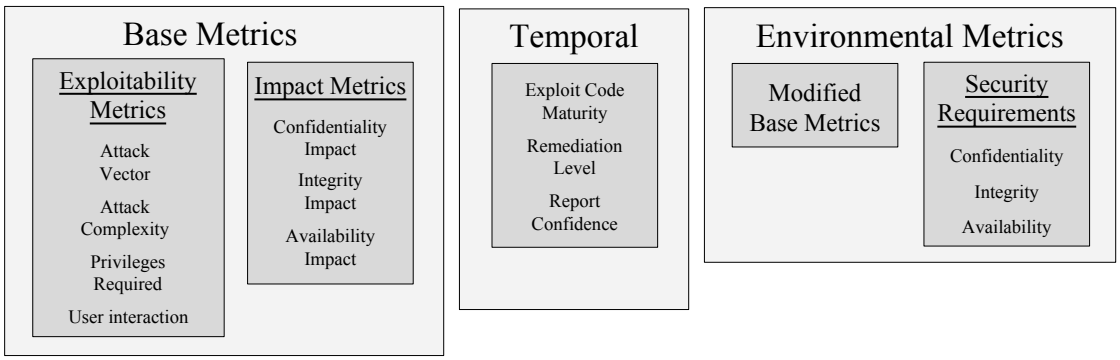

Figura 2.3: Esquema de grupos de métricas CVSS

Base Metrics

Estas métricas representam características fundamentais de uma vulnerabilidade que não mudam com o tempo.

Exploitability metrics: subgrupo de métricas que representam a dificuldade técnica associada à exploração de uma vulnerabilidade. As métricas que compõem este subgrupo são as seguintes:

- ***Attack Vector***: representa o nível de acesso físico necessário para explorar a vulnerabilidade. Por exemplo, se um *exploit* necessita de acesso físico ou local ao dispositivo ou pode ser executado remotamente.

- ***Attack Complexity***: métrica que descreve a necessidade da existência de condições fora do controlo do atacante para o sucesso de um ataque. Essas condições podem ser a necessidade de informação prévia sobre o sistema alvo, presença de configurações específicas, etc.

- ***Privileges Required***: associada à necessidade da existência de credenciais para a execução do ataque.

- ***User Interaction***: avalia a necessidade de uma interação, posterior ao ataque, por parte de um utilizador autorizado. Um exemplo é a existência de um *software* particular instalado pelo administrador do sistema.

Scope: métrica que representa a habilidade de uma vulnerabilidade num *software* afetar recursos para os quais o *software* vulnerável não tem privilégios de acesso. Um exemplo simples é a possibilidade de uma máquina virtual atacada ter acesso a recursos do sistema operativo anfitrião.

Impact metrics: subgrupo de métricas associadas ao impacto que um ataque com sucesso tem nas propriedades de segurança de um alvo. Para cada propriedade de segurança (Confidencialidade, Integridade, Disponibilidade) existe uma métrica definida.

Temporal Metrics

Métricas associadas à qualidade dos procedimentos de ataque e mitigação de uma vulnerabilidade num determinado momento. Representam características que podem facilmente mudar à medida que o tempo passa.

A primeira métrica (***Exploit Code Maturity***) está associada à maturidade do código dos *exploits* disponíveis para explorar a vulnerabilidade.

Na segunda métrica (***Remediation Level***) é avaliado o nível das remediações existentes da vulnerabilidade, ou seja, o estado e qualidade destas. Normalmente esta está associada à existência de um *patch* temporário ou oficial para a vulnerabilidade.

A última métrica (***Report Confidence***) é a confiança no relato da vulnerabilidade que representa o grau de confiança na sua existência e nos detalhes técnicos apresentados sobre esta.

Environmental metrics

Estas métricas permitem personalizar a pontuação CVSS consoante as especificidades do ambiente onde esta se encontra a ser calculada, afetando o conjunto de métricas *Base*. Mais especificamente, permitem pontuar uma vulnerabilidade contabilizando a importância do ativo na organização e considerando mecanismos de segurança alternativos ou adicionais não refletidos na pontuação. Consequentemente, estas são definidas por responsáveis pela gestão de ativos, visto que estes têm a informação sobre os ativos e os mecanismos de segurança aplicados a estes. Estão divididas em dois grupos, o grupo *Security Requirements Metrics* e as *Modified Base Metrics*.

- ***Security requirements metrics***: Conjunto de métricas que permitem afetar as métricas *Base* relacionadas com o impacto (Confidencialidade, Integri-

dade, Disponibilidade) tendo em consideração os requisitos específicos de um ativo. Para cada métrica de impacto é criada uma métrica modificada que representa a junção das duas métricas, *Base* e *Environmental*. Por exemplo, com esta métrica podemos evidenciar a necessidade de Confidencialidade num ativo onde uma vulnerabilidade apenas tem um impacto médio para este atributo.

- ***Modified Base Metrics***: Permitem o ajuste das métricas *Base* em relação a alterações feitas no ambiente que afetam a segurança do sistema, ou seja, afetam as métricas de *Exploitability*, *Impact* e *Scope*. Representam mecanismos de mitigação que não são considerados na pontuação *Base*.

Pontuação CVSS

A pontuação é representada por um valor que se situa entre 0.0 e 10.0, sendo que quanto maior o valor maior o risco de uma vulnerabilidade. O cálculo desse valor é feito com base numa fórmula que utiliza as métricas *Base*. Se forem definidas métricas (opcionais) temporais e ambientais (*Temporal* e *Environmental*) então podemos usar estas de forma a refinar o cálculo da pontuação. A pontuação tem o nome de *CVSS Score*.

Aquando do cálculo da pontuação é fornecido um *Vector String* que contém uma representação textual (siglas) das várias métricas que originaram a pontuação.

É ainda definida uma representação qualitativa da pontuação descrita na Tabela 2.1

Rating	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Tabela 2.1: Classificação qualitativa CVSS

2.8 *Vulnerability scanners*

Vulnerability scanners são ferramentas cuja missão principal é a **descoberta de vulnerabilidades** num conjunto de ativos presentes na rede[13][22][21][16]. Geralmente possuem uma base de dados de verificadores de vulnerabilidades. Estes verificadores definem-se por um conjunto de procedimentos envolvendo um ou mais pedidos a um ativo, sendo a descoberta da vulnerabilidade normalmente avaliada consoante a resposta do ativo.

Um verificador pode ser algo simples que verifique a versão de um determinado *software*, ou algo mais complexo como um pedido especialmente criado para perturbar o funcionamento de um ativo. Por isso é necessário ter algum cuidado a escolher o tipo de verificadores a serem usados, visto que uma configuração que faça testes que emulam ataques DDoS em ativos que se encontrem em produção pode perturbar o funcionamento de um ou mais serviços.

As bases de dados destas ferramentas, tipicamente possuem um grande número de verificadores o que faz com que um *scan* completo gere um grande número de pedidos e, como consequência, este tem um grande peso na rede e nos ativos testados, fazendo com que estes possam ser detetados como um ataque à rede.

Normalmente estes *scanners* permitem afinar aspetos relacionados com o desempenho dos *scans*, sendo os principais, o número de ativos a testar em simultâneo, número de verificadores a executar em simultâneo e o número de pacotes por segundo enviados pela ferramenta. Estes aspetos permitem ajustar o desempenho da ferramenta em relação às restrições de *hardware* da máquina e da rede onde esta realiza os *scans*. O ajuste desses aspetos também pode servir para evitar que a ferramenta seja bloqueada por *firewalls* ou sistemas de deteção de intrusões.

Um aspeto importante no planeamento do uso deste tipo de ferramentas é a conetividade entre a ferramenta e os seus alvos. Numa rede com um grande número de ativos, sub-redes e mecanismos de segurança, é necessário um planeamento rigoroso de forma a manter esta. Por exemplo, no caso de *firewalls*, pode-se permitir o tráfego que tem origem na máquina que está a correr a ferramenta ou então posicionar a ferramenta na sub-rede interior à *firewall*.

O resultado de um *scan* é um relatório onde normalmente são apresentadas todas as vulnerabilidades a que os alvos são suscetíveis, bem como o grau de severidade e informações adicionais destas. Neste relatório ainda podem ser especificadas outras informações, mediante configurações personalizadas, como os portos abertos, serviços ativos, pastas partilhadas, gráficos representativos, etc. É ainda importante referir que este tipo de ferramentas pode gerar falsos positivos, ou seja, são descobertas vulnerabilidades que na realidade não existem. Cabe ao utilizador desta ferramenta a tarefa de a investigar e experimentar de forma a ter a experiência necessária para detetar falsos positivos.

Este tipo de ferramentas normalmente possui uma consola de gestão onde os administradores e gestores de segurança da rede podem lançar *scans*, criar configurações de *scan*, adicionar alvos, gerar relatórios, etc.

Este tipo de ferramentas geralmente percorre os 4 passos apresentados na Figura 2.4, de forma a realizar um *scan*.

Scanning: processo inicial do *scan* que permite saber se um conjunto de alvos é alcançável e para cada alvo saber quais os portos abertos (TCP e UDP)[39][13][22].

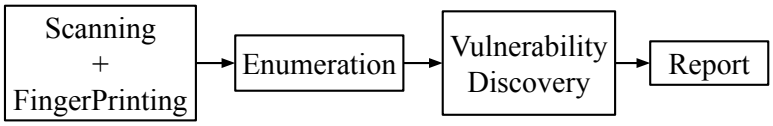

Figura 2.4: Fases da ação de um vulnerability scanner

Ainda é usada uma técnica complementar a esta, chamada ***Fingerprinting***, que permite descobrir o sistema operativo e as aplicações usadas pelo alvo, bem como as suas versões.

Enumeration: procedimento que permite obter informações relativas a recursos do alvo, como por exemplo serviços, aplicações, partilhas de recursos na rede, contas de utilizadores e respetivos privilégios, etc[13][22][5].

Estas duas fases permitem reduzir o número de verificações necessárias a executar, pois só vale a pena verificar vulnerabilidades sobre os vetores de ataque encontrados nestas fases.

Vulnerability Discovery: fase onde são executadas as verificações de vulnerabilidades sobre os alvos a testar, ou seja, são feitos os pedidos aos alvos, analisadas as respostas e consequentemente guardados os resultados caso seja pertinente[13][22].

Report: fase onde é gerado o relatório do *scan* realizado de acordo com os resultados encontrados[13][22]. Nesta fase a ferramenta pode interagir com sistemas de gestão de vulnerabilidades, gestão de incidentes e sistemas de *ticketing*.

De seguida são apresentados alguns dos *vulnerability scanners* mais conhecidos foram utilizados no projeto a desenvolver.

2.8.1 *OpenVAS*

Open Vulnerability Assessment System é uma *framework* que integra várias ferramentas e serviços com o objetivo de descobrir e gerir vulnerabilidades[28][13][32].

Na figura 2.5 é apresentada uma imagem com a arquitetura geral da ferramenta.

O módulo ***OpenVAS Scanner*** é responsável pela atividade de descoberta de vulnerabilidades. Para isso executa um conjunto de vários **NVT** (*Network Vulnerability Tests*) que se encontram armazenados na ferramenta, tipicamente um por vulnerabilidade. Estes são atualizados diariamente através de *feeds* presentes *online*.

OpenVAS Manager é o centro nevrálgico da ferramenta, sendo neste que podemos gerir todas as funcionalidades desta, ou seja, permite gerir todos os aspetos relacionados com *scans*. Este transforma o *OpenVAS* numa ferramenta gestão de vulnerabilidades. Comunica com os *scanners* através do protocolo ***OpenVAS Transfer Protocol*** (OTP) e oferece uma interface, baseada em XML, de interação com clientes chamada ***OpenVAS Management Protocol*** (OMP). Esta permite que vários tipos de clientes possam interagir com a ferramenta. Ambos os protocolos

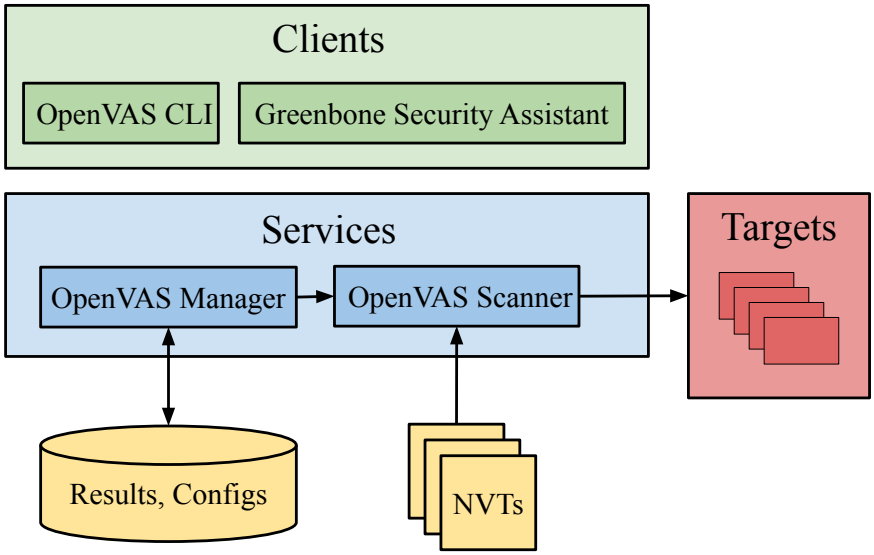

Figura 2.5: Arquitetura OpenVAS

utilizam SSL (*Secure Sockets Layer*) de forma a proteger as comunicações. Todos os resultados de *scans* e configurações são guardadas numa base de dados SQL. Ainda no *OpenVAS Manager* podemos gerir os utilizadores da ferramenta controlando o seu nível de acesso, designando-lhes grupos e papéis.

O *OpenVAS* fornece dois tipos de clientes, uma interface linha de comandos (***OpenVAS CLI***) que permite criar tarefas em *batch* e o ***Greenbone Security Assistant*** que fornece uma interface *Web* para *browsers*.

De seguida descrevem-se as principais funcionalidades fornecidas pelo *OpenVAS*.

- Agendamento de *scans* periódicos ou esporádicos.
- Vários métodos de descoberta de alvos que podem ser afinados e configurados.
- Permite usar e criar configurações de *scan* de vários tipos de vulnerabilidades, como por exemplo *Web*, sistemas, rede, etc. Estas configurações resultam na escolha dos NVTs (***Network Vulnerability Tests***) usados nos *scans*. Permite ainda a realização de *scans* autenticados nos alvos, de forma a minimizar falsos positivos, mediante o fornecimento prévio das credenciais do alvo.
- Definição de *Overrides* de forma a gerir resultados que contenham falsos positivos.
- Modo *Master-Slave* que permite criar um sistema distribuído de *scanners*. Um *OpenVAS Manager* é configurado como *master* e controla outros *Managers* que são *slaves*, sendo o *Master* responsável pelas atualizações de *software* e NVTs dos *Slaves*. Este modo permite realizar *scans* em zonas não acessíveis à

rede, protegidas por *firewalls*. Adicionalmente permite melhorar o desempenho de um *scan* pois distribui a carga dos *scans* em várias máquinas e permite diminuir a distância entre *scanners* e alvos, melhorando a ligação de rede entre estes.

- Produção de relatórios personalizados em vários formatos definidos por *plugins* (XML, HTML, LateX, PDF, etc.). Ao armazenar relatórios de vários *scans* é possível perceber a tendência de vulnerabilidades nos alvos. Para além disso o *OpenVAS* tem uma funcionalidade chamada *Delta Reports* que permite comparar os resultados de dois *scans*.

2.8.2 *Nexpose*

O *Nexpose* é uma solução completa de gestão de vulnerabilidades da *Rapid7*[30][22][32]. Esta acompanha o ciclo de vida de uma vulnerabilidade num sistema. Realiza a descoberta, deteção, verificação e mitigação de vulnerabilidades. Permite ainda a classificação de risco, análise de impacto e geração de relatórios.

Para interagir com este software é disponibilizada uma interface *Web* chamada ***Security Console*** e ainda uma ***API Web*** baseada em pedidos HTTP, ambos comunicam seguramente com o cliente usando SSL.

Nesta ferramenta um ativo tem a designação de ***asset***. Podem-se organizar estes *assets* em grupos e designar *tags* que os categorizem. Para realizar um *scan* é necessário criar um ***site*** que é um grupo de um ou mais *assets*, sobre o qual ou quais um *scan engine* ou um conjunto de *scan engines* (*scan engine pool*) irá realizar *scans* de vulnerabilidades tendo em conta uma configuração de *scan* designada *scan template*.

Um ***scan engine*** é o elemento fulcral do *Nexpose* que realiza um ou mais *scans* simultaneamente e no fim destes envia o resultado para a *Security Console*. Durante a sua operação guarda um *log* que permite acompanhar as suas ações individuais. Também podem ser formados conjuntos de *scan engines* o que permite melhorar o desempenho dos *scans*, torná-los tolerantes à falta de uma *scan engine* e ainda permite fazer *scans* em zonas protegidas por *firewalls*. Um *scan* é basicamente a execução de um conjunto de ***vulnerability checks*** que não são mais que testes a vulnerabilidades. Por sua vez um ***scan template*** é uma configuração que engloba um conjunto destes *vulnerability checks*. Existem *templates* já definidos ou podem ser criados novos *templates* personalizados.

Um requisito para criar um *site* é haver um ou mais ativos. Estes podem ser adicionados na *Security Console*, na forma de *hostnames*, IPs ou IP *ranges*, ou então pode ser feita uma descoberta dinâmica onde é fornecida uma ligação a um gestor de *assets* da rede que providencia essa informação (AD/LDAP, WinRM, servidores

DHCP).

Quando um *site* está definido podemos iniciar um *scan* sobre este. Inicialmente é feito um *scan* de descoberta, baseado na ferramenta *Nmap*[26], que permite verificar se os ativos alvo estão ativos e descobrir informações sobre estes. Com base nestas informações são escolhidos os *vulnerability checks* a executar no *scan* de vulnerabilidades. É possível realizar *scans* autenticados nos ativos o que melhora o resultado dos *scans*, sendo que é necessário introduzir as credenciais previamente na *Security Console*.

O *Nexpose* permite agendar *scans* e definir períodos chamados *blackouts*, onde não devem ser feitos *scans* de forma a aliviar os ativos e a rede em períodos críticos. Para além de vulnerabilidades, o *Nexpose*, ainda pode verificar se uma política de segurança está a ser aplicada corretamente, ou seja, se um conjunto de regras de segurança está a ser cumprido. Existem algumas políticas de organizações já definidas e pode-se criar políticas personalizadas baseadas nas já existentes.

É na *Security Console* que é feita a gestão dos utilizadores e as suas permissões. Esta gestão é importante para o acesso e distribuição de relatórios e também para a criação de *Exceptions*. Estas são definidas com o objetivo de ocultar resultados falso positivos nos relatórios.

O *Nexpose* tem ***templates* de pontuação de risco** e ainda permite personalizar estes. É possível ainda analisar tendências entre *scans*.

Os relatórios gerados podem ser totalmente personalizados tendo em conta os requisitos das organizações e é possível exportá-los numa grande quantidade de formatos. É possível definir, num *site*, um relatório *baseline* e também gerar um relatório que permita perceber a evolução e as tendências desse *site*, tendo em conta o número de ativos e vulnerabilidades.

Existe a possibilidade de interagir com outra ferramenta da *Rapid7*, o *Metasploit Pro*, com o objetivo de verificar as vulnerabilidades encontradas. O *Nexpose* permite que seja executado um *exploit*, caso exista um módulo desse no *Metasploit*.

2.9 Plataformas utilizadas

Um dos requisitos deste projeto é que os resultados dos *vulnerability scanners* fossem integrados nas plataformas existentes na DCY, de forma a enriqueceram estas com informação sobre vulnerabilidades nos ativos. Isto tem como objetivo melhorar a visão geral de segurança nos ativos de forma a avaliar melhor os níveis de qualidade de proteção (QoP) destes.

2.9.1 *ArcSight*

O ArcSight é uma solução SIEM (*Security Information and Event Management*), pertencente à HP, que tem como objetivo a análise eficiente de uma grande quantidade de dados de segurança (*Big Data Analytics*), de forma a auxiliar a proteção de ativos [1]. Esta solução recolhe dados provenientes de vários tipos de ativos (servidores, *routers*, PCs, etc.) na rede e, correlaciona e analisa estes. Os dados podem ser *logs* ou eventos. Desta forma consegue detetar ameaças e definir prioridades sobre estas, gerir atividades de resposta a incidentes e simplificar atividades de auditorias e processos de conformidade. Pode ser configurado para gerar alarmes caso ocorra um problema mais grave de segurança. A equipa da DCY utiliza este sistema para detetar problemas imediatos nos ativos críticos da PT (Altice Portugal).

2.9.2 Hidra

O Hidra (*High Performance Infrastructure for Data Research and Analysis*) é uma plataforma de análise de dados relativos a eventos de segurança. É usada pela DCY para análise estatística de eventos de segurança e geração de gráficos que permitem criar uma visão geral do estado de segurança dos ativos e a sua evolução. Com base nesta informação são gerados indicadores e *dashboards* para os vários *stakeholders*, desde as equipas operacionais que gerem os sistemas, até ao CTO e à própria administração da PT (Altice Portugal). O funcionamento do Hidra assenta na integração de 3 tecnologias base: *RabbitMQ*, *Elasticsearch* e *Kibana*.

RabbitMQ

O *RabbitMQ* é um *software message broker* que concretiza o protocolo *Advanced Message Queuing Protocol* (AMQP) [19]. Tem como objetivo providenciar às aplicações um mecanismo de envio de mensagens com fiabilidade, segurança, roteamento, orientação e sistema de filas. Existem clientes concretizados numa grande quantidade de linguagens de programação.

O Hidra utiliza esta tecnologia para gerir a receção de grandes quantidades de eventos, de diversas fontes, de forma fiável, eficiente e segura.

Elasticsearch

Elasticsearch é um motor de pesquisa de documentos de texto em notação JSON, baseado no *Apache Lucene* [9]. É distribuído, pode correr em múltiplas instâncias e permite a pesquisa e análise de dados em tempo real. O conteúdo dos documentos JSON é indexado o que permite uma pesquisa eficiente de documentos de texto. Os índices podem ser divididos em *shards* o que permite distribuir os índices, levando a

elasticidade de desempenho na procura e tolerância a perdas de índices. O *Elasticsearch* é usado no Hidra para pesquisar e filtrar, de forma eficiente, dados de eventos armazenados através da indexação de informação presente nestes.

Kibana

Kibana é uma plataforma *open-source* de análise e visualização de dados criada para trabalhar com *Elasticsearch* [17]. Permite procurar, visualizar e interagir com dados presentes em índices *Elasticsearch*. Os dados podem ser visualizados em tabelas, gráficos ou mapas. Fornece uma interface *Web* que permite criar *dashboards* dinâmicos, com os elementos de visualização disponíveis, que refletem em tempo real as pesquisas feitas no *Elasticsearch*.

Capítulo 3

Vulnerability Assessment Coordinator

Neste capítulo é feita uma descrição do problema tratado no projeto e dos requisitos para a criação do *Vulnerability Assessment Coordinator* (VAC). Seguidamente é apresentada a arquitetura da solução desenvolvida e uma breve análise sobre pormenores relacionados com o ambiente onde o VAC vai operar.

3.1 Descrição do problema

A DCY é responsável pela segurança de um grande número de ativos críticos. Uma vertente importante na proteção desses ativos é a descoberta e mitigação de vulnerabilidades que possam existir no seu *software*, visto estas, possibilitarem a ocorrência de ataques com sucesso comprometendo a sua segurança. Para que a DCY consiga cumprir a sua missão com sucesso, é essencial existir um processo de gestão de vulnerabilidades organizado e eficiente. Este deve contemplar a criação de uma visão geral e atual, ao longo do tempo, do estado de segurança de todos os ativos em termos de vulnerabilidades. Esta visão tem como base a monitorização e medição periódica do nível de vulnerabilidades que deve ser realizada em tempo útil. Deve também ser feita uma análise com o objetivo de caracterizar os ativos em termos da sua criticidade. Com isto é possível definir prioridades entre os ativos de forma a orientar uma estratégia de ação corretiva para as vulnerabilidades que estes possam ter. Neste momento a DCY tem uma equipa que trabalha num sistema de descoberta e mitigação de vulnerabilidades em aplicações Web. No entanto a avaliação de vulnerabilidades ao nível de serviços, sistemas e elementos de rede é algo que é feito manualmente. Tendo em conta o elevado número de ativos, que é preciso monitorizar, é difícil obter resultados em tempo útil que permitam que sejam efetuadas ações corretivas atempadamente. Para além disso os resultados dos *scans* que são feitos não estão a ser enviados para as plataformas usadas na DCY, o que faz com que estes sejam desperdiçados na medida em que podiam ajudar a detetar ocorrências ilícitas.

No caso do *ArcSight*, que é utilizado pelas equipas operacionais da DCY para efeitos de alarmística, os resultados de *scans* de vulnerabilidades podem enriquecer o conhecimento sobre ativos, o que melhora o funcionamento dos alarmes gerados. No caso do *Hidra*, a plataforma utilizada para investigação e análise forense, o armazenamento dos resultados permite que se possa fazer vários tipos de análise ao conjunto de ativos a gerir, como por exemplo acompanhar a evolução do nível de vulnerabilidade dos ativos, gerar *dashboards* e relatórios gerais ou individuais, cruzar dados de vulnerabilidades com outras informações na investigação de casos ilícitos, etc. A transformação e integração manual de resultados nestas plataformas é uma tarefa que iria consumir muito tempo e recursos humanos. Adicionalmente, num contexto de grande escala, a gestão de recursos relativamente às ferramentas de *vulnerability scanning* em função da disponibilidade dos ativos é algo que é complexo e moroso de ser feito manualmente, pelo que se torna necessário fazer um planeamento rigoroso e cuidado.

3.2 Requisitos

Os problemas apresentados na secção anterior levaram a que fosse proposta a criação de uma aplicação que permitisse automatizar a parte repetitiva do processo de gestão de vulnerabilidades, ou seja, o agendamento e execução de *scans* e a integração de resultados. Essa aplicação tem como base a orquestração e integração automática das varias ferramentas e plataformas utilizadas na DCY de forma a simplificar o processo de gestão de vulnerabilidades. A essa aplicação foi dado o nome de *Vulnerability Assessment Coordinator* (VAC).

Como referido anteriormente, a monitorização de vulnerabilidades de um ativo tem como base *scans* de vulnerabilidades periódicos que permitem avaliar e acompanhar o estado de segurança de um ativo ao longo do tempo. Esta é uma funcionalidade que a solução tem que realizar de forma automática, ou seja, a aplicação deve possibilitar ao utilizador fazer o agendamento de *scans* que serão despoletados e cujos resultados serão posteriormente integrados, de forma automática, sem intervenção do utilizador.

A solução deve possibilitar o agendamento de *scans* periódicos bem como a realização de *scans* espontâneos para lidar com situações específicas que possam acontecer, como por exemplo, a descoberta rápida de uma vulnerabilidade crítica cuja divulgação tenha acabado de ocorrer.

O processo de gestão de vulnerabilidades deve ter em conta a criticidade dos ativos a gerir para desta forma saber quais os momentos em que a operação destes é mais critica. Desta forma é possível planear o agendamento de *scans* de forma a ter o mínimo de impacto possível na operação dos ativos. Por esta razão o agendamento

da solução deve permitir configurar limites temporais de forma a impossibilitar a execução de *scans* nas janelas temporais críticas de operação dos ativos.

Deve existir uma interface que permita aos utilizadores e administradores interagir com a solução. A interface com o utilizador deve fornecer um conjunto de funções relacionadas com o agendamento de *scans* que encapsulem o máximo possível a complexidade de configuração e as especificidades associadas às ferramentas de *vulnerability scanning*. Cabe ao administrador da solução configurar as ferramentas de *vulnerability scanning* e as plataformas de armazenamento de resultados e, seguidamente, configurar a solução de forma que os vários elementos possam interoperar corretamente através da aplicação. Essa configuração deve ser feita através de uma interface de gestão desenvolvida para a aplicação.

Ao longo do tempo as necessidades da DCY podem levar a que seja necessário adicionar ou alterar tecnologias de *vulnerability scanning* para além das utilizadas de momento. Para que o VAC faça parte integral do processo de gestão de vulnerabilidades deve acompanhar facilmente essas mudanças e, para que isso aconteça, deve ser independente tanto quanto possível, de tecnologias concretas de *vulnerability scanning*. A alteração de tecnologias de *vulnerability scanning* deve, preferencialmente, ser feita sem afetar a disponibilidade da aplicação. No contexto de grande escala em que este projeto se enquadra é essencial que a aplicação consiga escalar com o aumento do número de ativos a gerir, bem como com o número de *vulnerability scanners* disponíveis. Assim, o VAC deve suportar várias instâncias de tecnologias de *vulnerability scanning* e, deve também fazer uma distribuição de carga de forma automática entre estas instâncias facilitando desta forma a gestão de recursos disponíveis.

Tendo em conta a importância e a longa duração dos *scans* é necessário que a aplicação consiga suportar falhas ao nível dos *scanners* e da própria aplicação. Desta forma o processo de *scans* não é interrompido devido à falha de um *scanner* nem a informação de agendamento é perdida caso a aplicação falhe.

A realização de *scans* com diferentes tipos de tecnologias de *vulnerability scanning* num ativo pode ser benéfica pois os resultados podem variar de maneira a que uma tecnologia descubra algumas vulnerabilidades que outra não descubra e vice versa[21]. A utilização de várias tecnologias permite que se obtenha um conjunto de resultados maior através do cruzamento de resultados, mas deve ser feito de forma ponderada tendo em conta a carga que os *scans* têm na rede e no ativo. Tendo isto em consideração, é necessário definir uma estratégia de utilização eficiente (em termos de custos monetários) das duas ferramentas de *vulnerability scanning* disponíveis. Neste momento as tecnologias utilizadas na DCY são o *Nexpose*, que necessita uma licença com um custo monetário associado, e o *OpenVAS*, que é uma ferramenta com uma licença publica que não tem custos para o utilizador, sendo

que o VAC deve trabalhar com estas duas tecnologias.

Desde o momento em que o utilizador agenda um *scan*, a solução deve realizar todos os passos necessários de forma automática para realizar os *scans* e integrar os resultados nas plataformas necessárias, ou seja, a solução é responsável pela execução dos *scans* no tempo definido no agendamento pelo utilizador, pela comunicação com as ferramentas de *vulnerability scanning* de forma a iniciar os *scans*, pela recolha dos dados quando os *scans* terminam e pela integração dos resultados nas plataformas pretendidas, *Hidra* e *ArcSight*.

Para suportar possíveis mudanças de plataformas de armazenamento de resultados e alterações nas tecnologias de *vulnerability scanning*, a funcionalidade de integração deve ser desenhada para ser modular de forma a conseguir transformar resultados provenientes de várias tecnologias de *vulnerability scanning* e enviar para várias plataformas de armazenamento de resultados. Preferencialmente estas mudanças não devem afetar a disponibilidade do VAC.

O VAC deve ainda ter uma funcionalidade que permita fazer a gestão de resultados incorretos (falsos positivos) de forma a que estes não sejam integrados nas plataformas de armazenamento de resultados.

É importante relembrar que estão fora do âmbito deste projeto vulnerabilidades de aplicações Web e *scans* com credenciais.

3.3 Arquitetura

Os *scans* de vulnerabilidades são compostos, em geral, por três fases de testes que são executadas sequencialmente. O teste de conectividade com o ativo, a descoberta de portos e serviços e, finalmente, os testes de vulnerabilidades. Quando estes terminam são gerados os resultados com base nas três fases de testes. A configuração do *scan* nas várias fases tem influência no tempo que este vai demorar a ser completado. Tipicamente uma configuração engloba dezenas de milhares de testes de vulnerabilidades mas são executados apenas um subconjunto que é definido pelo resultado dos testes de descoberta de portos e serviços. Dependendo do nível de segurança de um ativo e da configuração do *scan*, a lista de resultados pode ser grande. Outro fator que influencia o tempo dos *scans* são as condições de carga da rede e a disponibilidade de recursos nos ativos testados. Por isso pode-se perceber facilmente que um *scan* pode ser um processo pesado e moroso que pode durar várias horas.

Para a realização da gestão de vulnerabilidades é essencial fazer uma avaliação regular de segurança dos ativos envolvidos, o que torna essencial obter resultados de *scans* em tempo útil. O facto de este projeto se enquadrar num contexto de grande escala leva a que a arquitetura do VAC seja desenhada de forma a fazer uma gestão eficiente dos recursos, nomeadamente dos *vulnerability scanners* que são

os elementos que mais recursos consomem nesta arquitetura. Como mencionado anteriormente, a duração de um *scan* pode variar bastante pelo que a arquitetura deve ser desenhada de forma a isolar ao máximo a duração do processo de *scan*, ou seja, um *scan* mais prolongado não deve interferir com a conclusão e posterior integração de resultados de um *scan* que termine mais rapidamente.

A integração de resultados de um conjunto de ativos com muitas vulnerabilidades pode demorar algum tempo, visto que esta envolve a filtragem, transformação e envio dos resultados que tipicamente são enviados pela rede. O agendamento de *scans* deve ter o mínimo de interferência possível de forma a garantir o desencadeamento atempado dos *scans*.

Os factos apresentados anteriormente levaram a que fosse tomada a decisão de se dividir a arquitetura do VAC em três componentes centrais que representam as funcionalidades principais do VAC, tal como representado na Figura 3.1: Agendamento, Gestor de *Scanners* e Integração. Existe ainda o componente *Web* API que, como o nome indica, expõe uma API *Web* que permite aos utilizadores e administradores interagir com o VAC. A realização de um *scan* na plataforma VAC é composta por um fluxo de informação entre estes componentes, como também ilustrado na Figura 3.1.

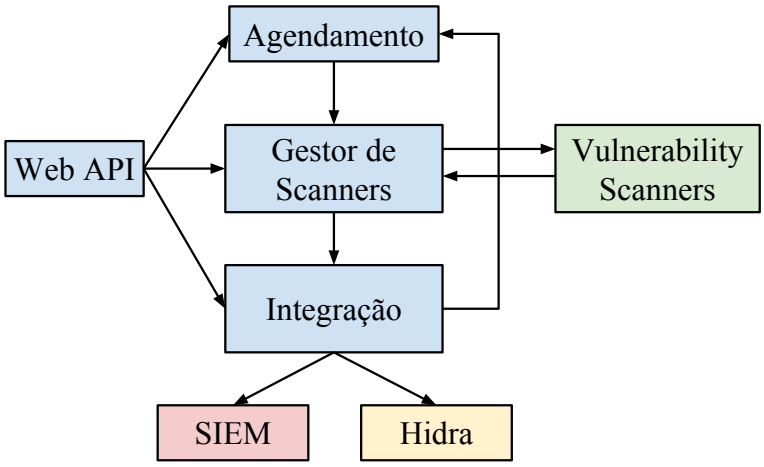

Figura 3.1: Arquitetura VAC

Seguindo o fluxo representado na figura 3.1, descreve-se de seguida, e de forma sucinta, o propósito dos diversos componentes e as interações entre eles.

Como referido anteriormente, o componente *Web* API expõe um conjunto de funções que permitem interagir com o VAC. Esta API tem como alvo dois grupos distintos: utilizadores e administradores. Os utilizadores têm funções relacionadas com a utilização normal da aplicação, ou seja, o agendamento de *scans*. Os administradores têm um conjunto de funções mais complexas ligadas à gestão do VAC e, das ferramentas e plataformas que este utiliza. Quando um utilizador quer agendar um

scan comunica com a *Web* API e esta comunica com o componente de Agendamento onde são definidos os parâmetros relativos ao *scan*.

O componente de Agendamento é responsável pelo desencadeamento atempado dos *scans* agendados pelos utilizadores conforme os parâmetros de temporização definidos.

As funções do componente Gestor de *Scanners* passam pela gestão das instâncias dos vários tipos de tecnologias de *vulnerability scanning* e pelo acompanhamento de execução de *scans*. Quando é recebida uma ordem de desencadeamento de um *scan* do componente de Agendamento, este componente escolhe uma instância de *vulnerability scanning*, comunica com essa instância para iniciar a execução do *scan*, espera a sua conclusão e extrai os resultados que deverão ser colocados num repositório. Este componente deve ainda encapsular a complexidade de configuração e interação com as ferramentas de *vulnerability scanning*.

O componente de Integração tem como responsabilidade consumir os resultados dos *scans*, que são enviados para o repositório pelo Gestor de *Scanners*, e enviá-los para as plataformas de destino. Para isso este componente deve ser capaz de transformar resultados nos formatos associados às tecnologias de *vulnerability scanning* nos formatos das plataformas que recebem os resultados. Ainda é responsável pela filtragem de resultados falso positivos.

De seguida é feita uma descrição mais detalhada de cada componente central do VAC.

3.3.1 Agendamento

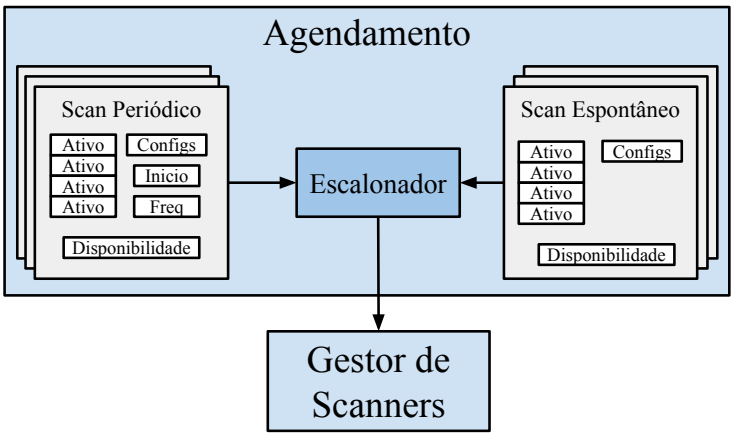

Figura 3.2: Componente Agendamento

O componente de agendamento (Fig. 3.2) tem como objetivo principal o desencadeamento atempado dos *scans*. Neste é armazenada uma coleção de *scans* que são agendados pelo utilizador. Existem dois tipos de *scans*: periódicos e espontâneos.

Os *scans* periódicos são *scans* que devem ser executados repetidamente segundo uma frequência, servindo para avaliar o estado de um ativo de forma regular. Os *scans* espontâneos servem para situações pontuais e urgentes em que é necessário executar um *scan* o mais rapidamente possível.

Estes *scans* partilham o seguinte conjunto de parâmetros:

- Coleção de ativos sobre os quais o *scan* se vai realizar e respetivas características.
- Janela diária de disponibilidade dos ativos.
- Configurações de *scan*.

O *scan* periódico tem os seguintes parâmetros de temporização do *scan*:

- Frequência com que o *scan* se realiza (diária, semanal, mensal).
- Data de início do *scan*.

De maneira a concretizar a estratégia de utilização de várias tecnologias de *vulnerability scanning* consoante o custo, neste componente um *scan* periódico por omissão utiliza a tecnologia primária, que tem menos custo monetário. No *scan* espontâneo existe um parâmetro definido pelo utilizador onde pode ser escolhida a tecnologia de *vulnerability scanning* a utilizar, visto estes serem um tipo de *scan* menos utilizado.

O subcomponente Escalonador tem como tarefa percorrer as listas de *scans* periodicamente e desencadear os *scans* consoante o tipo de *scan* e os parâmetros temporais que estejam definidos. Para desencadear um *scan* este subcomponente comunica com o componente Gestor de *Scanners* a ordem para desencadear o *scan*, fornecendo-lhe e os parâmetros associados a este. O *scan* apenas é desencadeado pelo componente de Agendamento se a janela diária de disponibilidade dos ativos o permitir.

3.3.2 Gestor de *Scanners*

O componente Gestor de *Scanners* (Fig. 3.3) é composto pelo subcomponente de Acompanhamento que gere todos os aspetos do *scan*, desde a sua criação, acompanhamento da sua execução nos *vulnerability scanners*, finalização e extração de resultados. O subcomponente *Scanners* é responsável pela gestão das várias instâncias de *vulnerability scanners* e por fazer o mapeamento entre *Templates* do VAC e configurações especificas de tecnologias de *vulnerability scanning*. Adicionalmente é responsável pela distribuição de carga entre as várias instâncias e por gerir a falha de uma instância de *vulnerability scanning*.

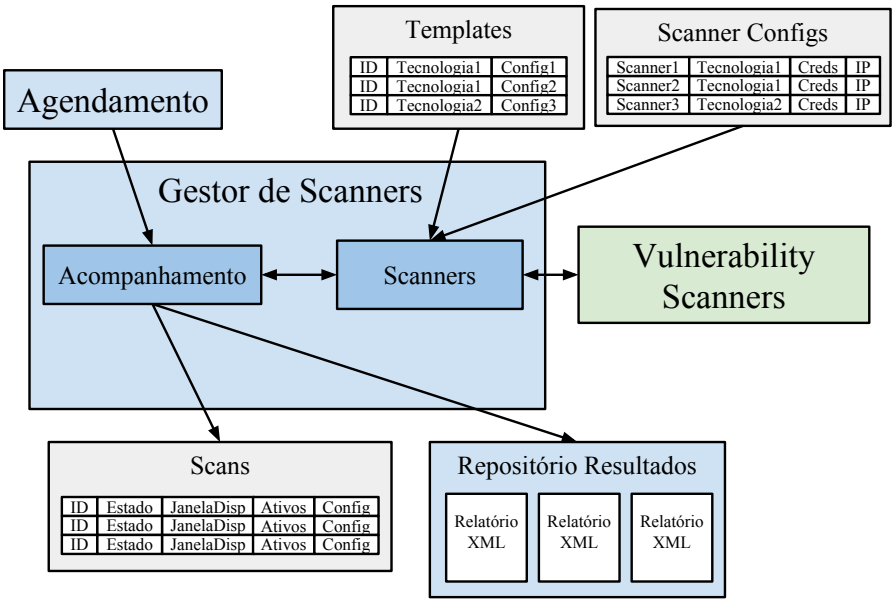

Figura 3.3: Componente Gestor de Scanners

Este é o componente que concretiza a abstração tecnológica do VAC através de módulos que sabem comunicar com os *scanners* das diferentes tecnologias de *vulnerability scanning* e com o mapeamento de *Templates* do VAC para configurações especificas das tecnologias.

Quando o componente de Agendamento envia uma ordem para despoletar um *scan*, comunica com o subcomponente de Acompanhamento que começa por criar uma instância de um *Scan*. Essa instância de *Scan* é uma representação de um *scan* que está a correr num *scanner* e é composta por:

- Um identificador relacionado com o *scan* que é executado no *vulnerability scanner*.
- O estado do *scanner* que varia entre *novo*, a *executar*, em *pausa* e *terminou*.
- A lista de ativos.
- A janela de disponibilidade dos ativos.
- Configuração do *scan*.

Quando a instância de *Scan* é criada o seu estado é colocado como novo e o identificador relacionado com o *scan* não é preenchido. Os restantes parâmetros são extraídos na comunicação com o componente de Agendamento. O subcomponente de Acompanhamento tem uma lista de instâncias de *Scan* pelas quais é responsável. Este percorre a lista periodicamente e realiza uma ação predefinida conforme o

estado em que este se encontre. De seguida é feita uma descrição das ações que o subcomponente de Acompanhamento realiza consoante o estado da instância:

- **Novo**: Comunica com o subcomponente de *Scanners* que, consoante o *Template* na sua configuração e a tecnologia de *vulnerability scanning* a utilizar, consulta a configuração específica associada à tecnologia de *vulnerability scanning*, escolhe a instância onde vai executar o *scan* e inicia a execução do *scan*. É neste passo que o identificador é preenchido com a referência para o *scan* dada pelo *vulnerability scanner*. Depois de a operação concluir com sucesso, o estado muda para a *executar*.

- **A executar**: Comunica com o subcomponente *Scanners* e este por sua vez consulta o *vulnerability scanner* que está a executar o *scan* para saber se este já terminou. Se a execução terminou muda o estado para *terminou*. Se a janela diária de disponibilidade dos ativos que estão a ser alvos de *scan* terminou é dada a ordem para o *scan* terminar e o seu estado muda para *terminou*. Caso o *scan* esteja a executar não é efetuada nenhuma ação.

- **Terminou**: Comunica com o subcomponente *Scanners* para extrair os resultados do *vulnerability scanner* e, posteriormente, guarda-os num repositório, tipicamente na forma de um relatório XML num formato associado à tecnologia utilizada. Adicionalmente dá a ordem para todos os dados referentes ao *scan* serem apagados no *vulnerability scanner*.

3.3.3 Integração

O componente de Integração (Fig. 3.4) é responsável por verificar periodicamente a existência de novos resultados no repositório. Quando estes existem são enviados para o subcomponente de Filtragem. Este componente, com base numa coleção de regras associadas a resultados falso positivos, elimina os resultados que correspondam a essas regras. De seguida os restantes resultados são enviados para o subcomponente de Integração onde serão transformados de acordo com a plataforma de destino.

Este componente é modular de forma a poder suportar o envio de resultados para várias plataformas. Neste momento existem os módulos de *ArcSight* e *Hidra*. No caso do *ArcSight* foi decidido que o formato de envio será um relatório XML baseado na tecnologia de *vulnerability scanning Nexpose*. Para o Hidra é feita uma transformação dos resultados em eventos com um conjunto de campos comuns às varias tecnologias de *vulnerability scanning* e os restantes campos específicos de cada tecnologia. Adicionalmente, os resultados são enviados para o módulo de Avaliação

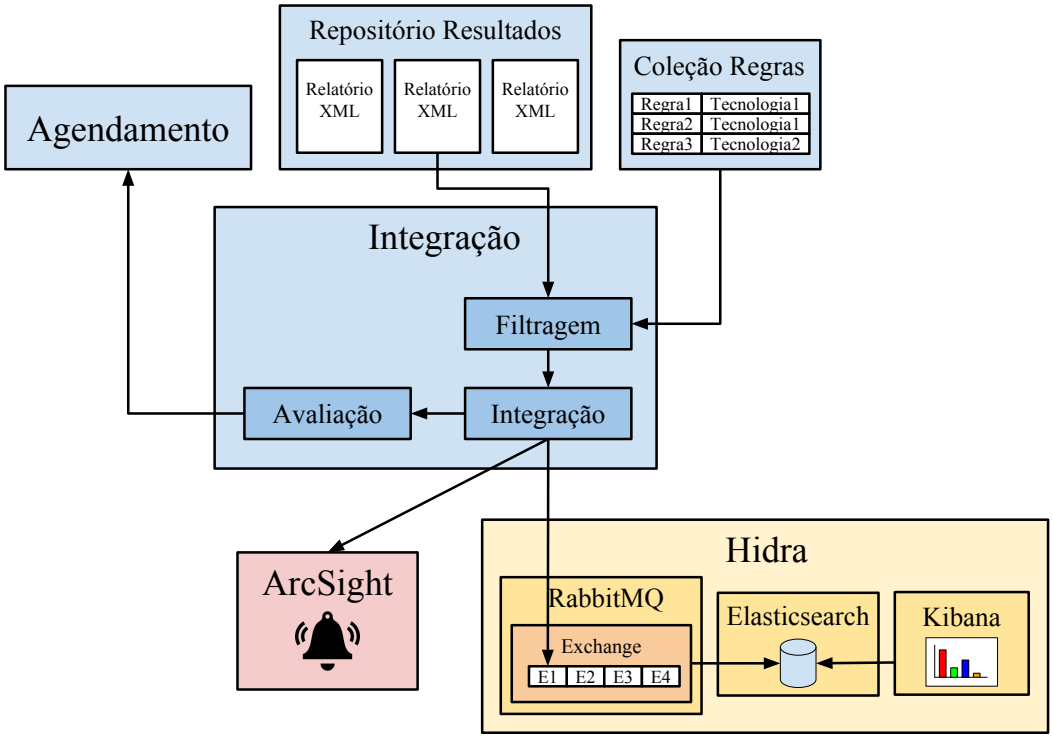

Figura 3.4: Componente Integração

que, com base numa heurística baseada nas características dos ativos pode executar um *scan* de segunda opinião de forma a reforçar e possivelmente melhorar a descoberta de resultados do primeiro *scan*.

3.4 Ambiente de operação

Como descrito anteriormente esta solução tem como objetivo integrar um conjunto de ferramentas e plataformas. Algumas já estavam em produção (*Hidra*, *ArcSight*, *Nexpose*), outras foram colocadas (*OpenVAS*, VAC). Ainda são parte integrante deste sistema os ativos que são alvos do *scans*.

Os vários elementos referidos anteriormente estão obviamente dispersos pela rede onde realizam as suas atividades. Visto que o tipo de operações realizadas pelo VAC depende da conectividade entre os vários elementos e tem um impacto significativo na rede, foi importante perceber como estes elementos estão interligados e a sua localização na rede.

Com base no estudo feito previamente das ferramentas e plataformas foi possível prever o tipo de interações que estas iriam ter com o VAC. Com base nessa análise foi possível identificar os principais desafios associados ao ambiente onde as ferramentas, plataformas e ativos operam, e desta forma desenhar o VAC para melhor lidar com estes.

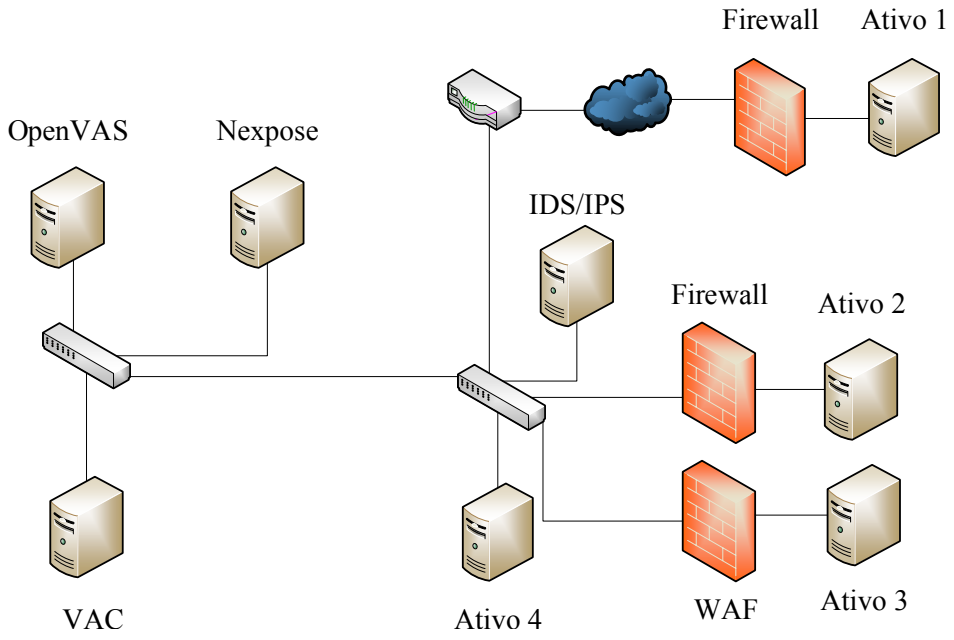

Figura 3.5: Ambiente de operação

A figura 3.5 representa uma possível configuração de rede com os vários elementos que fazem parte da solução. Em termos de conectividades, o VAC tem que ter conectividade com os *vulnerability scanners*, *OpenVAS* e *Nexpose*, com as plataformas que recebem resultados, *Hidra* e *ArcSight*, e com os seus utilizadores.

A interação do VAC com os *vulnerability scanners* consiste em pedidos de de-

sencadeamento de *scans*, consultas periódicas sobre o estado dos *scans* e extração de resultados de *scans*. Facilmente se percebe que vai existir um grande número de comunicações entre o VAC e os *vulnerability scanners* pelo que o desempenho do VAC, em especial do componente Gestor de *Scanners*, depende fortemente da qualidade da ligação de rede entre estes. O VAC também tem que ter conectividade com as plataformas que recebem os resultados, que neste caso são o *Hidra* e o *ArcSight*. O componente de Integração filtra e transforma os resultados e depois usa essas conectividades para enviar os resultados. No caso do *Hidra* os resultados são enviados um a um (eventos) e, para o *ArcSight*, vão aglomerados, tipicamente um relatório XML de um *scan*. O desempenho do VAC está intrinsecamente ligado ao desempenho dos *scans* pois estes são uma parte central da sua atividade, logo é importante analisar alguns fatores que influenciam o desempenho destes.

Como referido anteriormente, um *scan* de vulnerabilidades consiste em 3 fases: teste de conectividade, descoberta de portos e serviços e testes de vulnerabilidades. Se existir conectividade o *vulnerability scanner* vai fazer um *scan* de portos e, sobre os portos abertos que encontrar, vai realizar testes com o objetivo de descobrir serviços conhecidos. Com base nos portos e serviços que forem descobertos o *vulnerability scanner* vai fazer uma seleção dos testes de vulnerabilidades que tem disponíveis, tipicamente dezenas de milhares, e executar essa seleção.

Para um *scan* ser bem executado tem que existir conectividade entre o *vulnerability scanner* e o ativo alvo, ou seja, na rede e nos elementos que a compõem tem que ser assegurada ligação física e rotas configuradas corretamente. Cada teste de vulnerabilidade consiste em pelo menos um pedido feito pelo *vulnerability scanner* e por uma resposta do ativo, logo um *scan* vai gerar milhares de pedidos e respostas. Estes pedidos traduzem-se numa grande quantidade de tráfego que sobrecarrega os ativos que são testados e o troço de rede entre o *vulnerability scanner* e os ativos. Facilmente se percebe que o desempenho dos *scans* está relacionado com os recursos dos *vulnerability scanners*, os recursos dos ativos testados e a qualidade das ligações de rede entre estes. É importante perceber que a qualidade da ligação de rede entre os *vulnerability scanners* e os ativos está associada à largura de banda e latência entre estes. Alguns dos fatores que podem afetar esses atributos são:

- Número de elementos.
- Distância.
- Tipo de ligação física (cobre, fibra, sem fios).
- Rotas alternativas (que podem ser usadas em caso de congestionamento).

Outro fator que influencia o desempenho e a exequibilidade dos *scans* são os mecanismos de proteção existentes na rede. Este tipo de *scans* é caracterizado por

executar um elevado número de pedidos e interagir com um vasto número de portos. Este tipo de comportamentos é normalmente detetado como um ataque e bloqueado por mecanismos de proteção existentes na rede. Logo é importante ter em conta estas situações na realização de *scans* e perceber o impacto que estes mecanismos podem ter na realização de *scans*. De seguida são expostas algumas situações que podem ocorrer e possíveis soluções.

Uma *firewall* que proteja uma sub-rede pode bloquear pedidos do *vulnerability scanner*, o que poderá gerar falsos negativos (Ativos 1 e 2 na Figura 3.5), ou seja, os pedidos não chegam à máquina e uma possível vulnerabilidade não vai ser descoberta. Para além disso, o bloqueio de pedidos vai afetar o desempenho dos *scans* devido a um intervalo de tempo que os *vulnerability scanners* esperam antes de interromperem um teste, intervalo que é tipicamente definido nas configurações do *scan*. Existem duas maneiras de ultrapassar este problema. A primeira consiste em configurar a *firewall* de forma a permitir a passagem de todo o tráfego que tenha origem no *vulnerability scanner*. No entanto é importante perceber que esta solução pode trazer um problema de segurança visto que se um atacante descobrir que o endereço IP do *vulnerability scanner* não é analisado, pode forjar o endereço IP dos seus pacotes (IP *spoofing*) e realizar ataques sem ser detetado. Alternativamente pode ser colocado um *vulnerability scanner* dentro da rede protegida pela *firewall* e aberta uma ligação entre este e uma consola de gestão central (fora da sub-rede protegida pela *firewall*) que será responsável pela gestão deste. Desta forma, o *scanner* tem conectividade total para os ativos melhorando assim a precisão dos resultados obtidos e o desempenho do *scan*.

Um sistema de deteção e prevenção de intrusões analisa ligações e o conteúdo de pacotes na rede de forma a detetar e bloquear ligações que se encontrem a realizar ações suspeitas (Ativos 1 e 2 têm *firewalls* que podem ser configuradas pelo IDS/IPS na Figura 3.5). Os *vulnerability scanners* realizam muitos pedidos com conteúdo suspeito e sendo que a sua ação pode ser considerada semelhante a um ataque pode levar a que um sistema de deteção e prevenção de intrusões possa bloquear os pedidos de um *scan*. Afinar o *vulnerability scanner* de forma a limitar o número de pedidos por segundo é uma forma resolver este problema, no entanto esta solução diminui o desempenho dos *scans*. Outro modo de solucionar este problema passa por colocar o endereço IP dos *vulnerability scanners* numa *whitelist* dos IDS/IPS de forma a permitir que os *scans* possam ser feitos sem restrições. No entanto esta solução pode trazer problemas de segurança iguais aos mencionados anteriormente no caso da *firewall*.

Outro mecanismo muito usado em servidores Web são as WAF (*Web Application Firewall*), que analisam os pedidos recebidos e bloqueiam aqueles que possam ser ataques (Ativo 3 na Figura 3.5). Este mecanismo apresenta um comportamento si-

milar a um IDS/IPS, pelo que pode prejudicar o resultado dos *scans*. Este problema pode ser solucionando de forma semelhante ao caso dos IDS/IPS. Relembrando que o foco da solução é a descoberta de vulnerabilidades em redes e sistemas e não em aplicações Web, é importante referir que a descoberta de vulnerabilidades em servidores Web, especialmente nas suas configurações, é uma das tarefas da solução. Um teste de vulnerabilidades a servidores Web apresenta um comportamento semelhante a um ataque a aplicações Web, e, portanto, torna-se essencial considerar o impacto das WAF nos *scans* a realizar.

Idealmente todos os *scans* deveriam ser feitos com todas as conectividades abertas e boa qualidade de ligação entre o *vulnerability scanner* e os ativos, visto que desta forma existem condições para um *scan* encontrar o número máximo de resultados possível. No entanto, nem sempre é possível termos esse cenário, pelo que é importante perceber que a presença de mecanismos de proteção compromete a precisão dos resultados dos *scans*. A decisão de realizar *scans* com a influência dos mecanismos de proteção deve estar associada à confiança depositada na segurança destes. Efetivamente, se um atacante conseguir circunscrever esses mecanismos pode explorar uma vulnerabilidade não encontrada pelos *vulnerability scanners*.

Capítulo 4

Concretização

Neste capitulo é explicada, com algum detalhe, a concretização do VAC. Inicialmente são mostradas as afinações necessárias para tornar o processo de *scan* eficaz em tempo útil (4.1) e também algumas configurações feitas nas plataformas de armazenamento para que estas estejam preparadas para receber dados da aplicação (4.2). As secções 4.3 e 4.4 mostram alguns dos conceitos base em que o VAC se baseia. Na secção 4.5 são explicadas as decisões tecnológicas, a arquitetura de software e outros aspetos relacionados com o desenvolvimento da aplicação. As secções 4.6 , 4.7 e 4.8 apresentam respetivamente as funcionalidades de interface *Web*, tolerância a falhas e *log* do VAC. O funcionamento mais detalhado da aplicação é explicado na secção 4.9 onde é seguido o fluxo de agendamento, execução e integração de resultados ao longo dos vários componentes e classes da aplicação. As secções finais (4.10 e 4.11) mostram sucintamente o trabalho realizado para transformar os resultados no formato dos *vulnerability scanners* para as plataformas de armazenamento de resultados.

4.1 Afinação de *scanners*

Para ser possível fazer uma gestão eficaz de um elevado número de ativos é essencial obter resultados periodicamente em tempo útil, e com isto obter uma avaliação contínua do estado dos ativos a nível de vulnerabilidades e, a partir daí definir e executar um plano de ações corretivas atempadamente. Num ambiente de operação diverso que pode apresentar algumas características que dificultam a realização de *scans* (Secção 3.4) bem como uma configuração de *scan* bastante abrangente, pode levar a que alguns *scans* se prolonguem por horas ou dias até.

Idealmente seria desejável que os *vulnerability scanners* realizassem todos os testes de vulnerabilidades em todos os vetores de ataque remotos de um ativo, adaptando-se a todas as condições adversas possíveis no ambiente de operação. Este caso ideal normalmente aumenta de forma acentuada a duração dos *scans*. Uma

aproximação deste caso ideal são os testes de penetração onde não existe nenhuma informação prévia sobre os ativos. Neste tipo de testes o objetivo é encontrar o número máximo de vulnerabilidades de ativos de forma a comprometer a sua segurança, fazendo uma avaliação final sobre os problemas encontrados. Tipicamente os requisitos temporais deste tipo de testes não são muito restritos. No caso deste projeto é feita uma abordagem diferente, pois existem requisitos temporais mais restritos devido à necessidade de se obter resultados em tempo útil num ambiente de operações onde os recursos são partilhados por vários ativos. Nesse ambiente de operações existem alguns recursos que podem ser controlados e existe algum conhecimento prévio sobre o tipo de serviços e sistemas operativos utilizados pelos ativos a testar. Logo, é possível reduzir alguns vetores de ataque a explorar e testes a realizar pelos *vulnerability scanners* de forma a reduzir o esforço realizado nos *scans*. Também é possível afinar os parâmetros de temporização dos *scans* com mais certeza de forma a limitar a duração máxima dos *scans*. Quanto menores forem os valores de *timeout* dos testes menor o tempo total de *scan*. Em contrapartida existe uma maior probabilidade de alguns testes não serem realizados corretamente, especialmente nos casos em que o ambiente de operação apresenta condições adversas (congestão na rede, *firewalls*, IDS, etc.), levando a uma perda de precisão nos resultados. Logo, é importante ter cuidado com o processo de afinação das configurações de *scan* dos *vulnerability scanners*. Este deve ser feito com base numa escolha ponderada entre três fatores:

- Precisão.
- Duração.
- Recursos.

Estes fatores são interdependentes sendo que uma afinação que afete um ou dois fatores vai ter um impacto proporcional no(s) restante(s). É inviável uma afinação que beneficie todos os fatores. De uma forma genérica esta relação pode ser visualizada através da seguinte fórmula:

$$Precisão = Duração \times Recursos$$

Segundo esta fórmula se, por exemplo, existirem constrangimentos temporais que obriguem a que a duração máxima de um *scan* tenha que ser diminuída, deve-se aumentar os recursos ou a precisão irá ser afetada. De igual modo, se os recursos diminuírem a duração tem que aumentar para não afetar a precisão. Se num *scan* a precisão necessária for menor podemos diminuir a duração, os recursos utilizados ou ambos. Importante referir que o fator recursos se refere não só aos recursos da

máquina que realiza o *scan* mas também aos recursos do ativo que está a ser alvo do *scan*, bem como os recursos de rede existentes entre estes. Logo, as características do ambiente de operação são um fator muito importante no processo de afinação de configurações.

Tipicamente os *vulnerability scanners* adaptam-se às características do ambiente de operação alterando a sua parametrização no decorrer de um *scan*. Numa primeira fase tentam utilizar o máximo de recursos possível (dentro dos parâmetros inicialmente definidos) de forma a terminar o *scan* da forma mais rápida possível. Quando os recursos se esgotam, pela sua exaustão ou pela ação de mecanismos de proteção, os *vulnerability scanners* adaptam-se aumentando a duração do *scan*.

Os *vulnerability scanners* que vão ser utilizados pelo VAC encontram-se num ambiente de operação onde existem sistemas de produção pelo que as configurações utilizadas relativas a recursos vão ser as predefinidas para evitar que os *scans* causem perturbações nesses sistemas. Logo as afinações vão incidir sobre os fatores de precisão e duração.

Para fazer essa escolha foi preciso testar e perceber como funcionam os *vulnerability scanners* utilizados neste projeto. Relembrando também que o foco do projeto são vulnerabilidades ao nível de serviços, sistemas e elementos de rede, é abordada ainda a afinação das configurações de *scans* tendo em conta esta vertente. É importante também mencionar que *scans* com credenciais acrescentam grande precisão pois permitem que o *vulnerability scanner* aceda à máquina como utilizador e faça testes internos ao sistema operativo e ao software instalado. Estes aumentam consideravelmente a duração dos *scans* mas como não se encontram no âmbito do projeto não são considerados nas afinações.

4.1.1 *OpenVAS*

O *OpenVAS* é uma aplicação de *vulnerability scanning* que para realizar um *scan* coordena a execução de um conjunto *plugins* (NVTs - *Network Vulnerability Tests*), segundo uma determinada ordem, contra um conjunto de ativos alvo. Um *plugin* é um *script* que geralmente executa um conjunto de instruções ou um programa externo de forma a testar um alvo. No *OpenVAS* os *plugins* estão organizados em grupos chamados famílias de NVTs.

Esses grupos estão organizados pelo tipo de testes, sistemas, protocolos e aplicações que os *plugins* testam. Por exemplo existe uma família chamada *Port scanners* que tem um conjunto de *plugins* associados ao teste de conectividade e descoberta de portos no alvo, como também algumas famílias de *plugins* de descoberta de serviços. Existe ainda uma família para *Firewalls*, produtos *Cisco*, sistemas operativos *Windows* e várias distribuições de Linux, servidores FTP, protocolo SNMP, bases de dados, servidores web, abusos de aplicações *web*, etc. Uma configuração de *scan*

é uma escolha de quais os *plugins* a executar bem como algumas preferências que possam ser configuradas nesses *plugins*.

Quando um *scan* é executado, o *OpenVAS* executa os *plugins* da configuração por uma determinada ordem que segue os passos de execução de um *vulnerability scanner* (Fig. 2.4). Primeiro começa por executar *plugins* de teste de conetividade e, se o ativo estiver acessível, executa depois testes de descoberta de portos depositando os resultados deste teste numa *Knowledge Base* (KB), onde são guardados todos os resultados do *scan* enquanto este está em execução. De seguida executa os *plugins* de descoberta de serviços com base na informação encontrada sobre os portos abertos na máquina e guarda os resultados na KB. No próximo passo executa os restantes *plugins* para descobrir vulnerabilidades. Em grande parte dos casos os *plugins* de vulnerabilidades só executam caso o serviço e/ou porto existam na KB, caso contrario não são executados. No fim da execução dos *plugins* é criado um relatório do *scan* com base no resultado dos testes.

Nos testes iniciais feitos com o *OpenVAS* em máquinas localizadas na Internet foi observado que *scans* em alguns *hosts* ficavam bloqueados uma quantidade excessiva de tempo em 1% de progresso. Após ser feita uma análise dos processos do *OpenVAS* em execução, concluiu-se que o *scanner* se encontrava a executar a ferramenta *nmap*, que utiliza para fazer a descoberta de portos nos ativos. Este corre o *nmap* através de um *plugin* existente na família *Port scanners*.

O *nmap* é uma ferramenta utilizada para a descoberta de conetividade, portos abertos, sistemas operativos e serviços utilizados por máquinas numa rede[26]. O seu funcionamento na tarefa de descoberta de portos define-se sucintamente pelo envio de sondas para ativos que respondem, ou não, a estas sondas. O tipo de respostas ou falta delas determina o estado dos portos de um ativo em relação à máquina que envia a sonda. A velocidade de envio destas sondas é amplamente configurável, de forma a ser possível realizar testes muito rápidos e obter resultados rapidamente com menor precisão, ou realizar testes muito lentos que consigam ter sucesso na presença de *firewalls* ou IDSs.

De forma a perceber o atraso significativo neste *scan* foi feita uma captura dos pacotes enviados do *nmap* para os ativos e observou-se uma variação instável do intervalo de tempo entre o envio de sondas para o ativo, em alguns casos 500 ms. Tendo em conta a elevada duração dos *scans*, devido à situação exposta anteriormente, foi realizado um estudo da ferramenta *nmap*, com foco especial nos parâmetros relativos ao desempenho e temporização [25]. Após esse estudo foi encontrados um conjunto de fatores e parâmetros que afetam o desempenho e a duração máxima do processo de descoberta de portos do *nmap*, sendo estes apresentados de seguida.

- **Método de teste TCP** : representa o tipo de teste realizado a um porto TCP. Existem vários métodos de teste TCP mas para este trabalho apenas

são relevantes o *TCP SYN scan* e o *TCP connect scan*.

O *TCP SYN scan* (Fig. 4.1) realiza apenas 2 dos 3 passos do processo de estabelecimento de uma ligação TCP (*Three-Way Handshake*). Um pacote TCP, onde é definido o porto a testar, é enviado para o ativo com a *flag* SYN ativa, se este responder com um pacote com as *flags* SYN e ACK ativas significa que o porto está aberto. De seguida o *nmap* envia um pacote com a *flag* RST ativa para terminar a ligação. Ao não completar o *three way handshake* (Fig. 4.1 - Porto aberto) este método tem a vantagem de, em alguns sistemas, a sua atividade não ser registada. Caso o ativo responda ao primeiro pacote com outro pacote com a *flag* RST ativa, significa que o porto está fechado (Fig. 4.1 - Porto fechado). Este tipo de *scan* trabalha num nível mais baixo necessitando de acesso privilegiado, na máquina onde o *nmap* está, para poder manipular os pacotes diretamente o que permite que os testes sejam feitos de forma mais eficiente.

O *TCP connect scan* realiza o *Three-Way Handshake* completo (Fig. 4.1 TCP Connect Scan), ou seja mais um passo que o *TCP SYN scan*. Este método é usado normalmente quando não existe acesso privilegiado à máquina onde o *nmap* é executado e usa a chamadas ao sistema operativo para realizar os testes (nível mais alto que o *TCP SYN scan*).

Logo o método *TCP SYN scan* é mais eficiente visto ser executado num nível mais baixo e utilizar menos pacotes do que o método *TCP connect scan*.

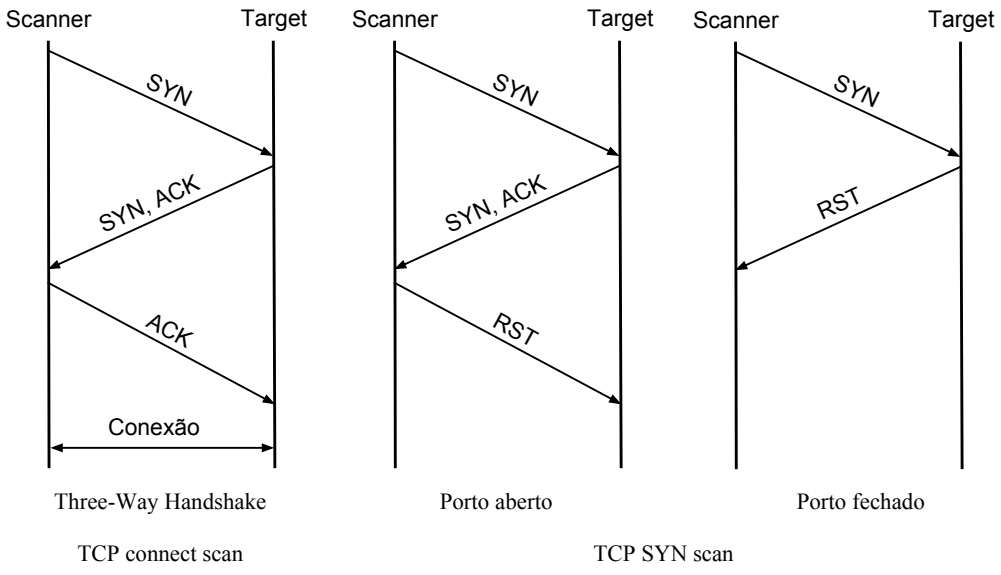

Figura 4.1: Tipos de scans TCP

- **Número de portos** - O número de portos a testar é o fator mais importante relativamente à duração do *scan*. Quanto maior o número de portos maior vai

ser o número de testes a realizar e consequentemente maior o tempo total da descoberta de portos.

- ***Max Scan Delay*** - este parâmetro define o tempo de espera máximo entre o envio de sondas, ou seja tentativas de ligação.
- ***Max RTT Timeout*** - parâmetro associado ao *round trip time* de um pacote que define o tempo de espera máximo até se considerar que uma tentativa de ligação falhou.
- ***Max Retries*** - numero de tentativas de ligação até se considerar que um porto não está aberta.
- ***Max Rate*** - parâmetro que controla o número máximo de pacotes enviados por unidade de tempo.
- ***Min Parallelization*** - numero de sondas ativas simultaneamente.

A maioria dos parâmetros relativos a tempo e desempenho podem ser definidos com um valor inicial e valores limite (máximo e mínimo). Quando o *scan* inicia o *nmap* utiliza os parâmetros iniciais definidos. Caso sejam detetadas perdas nas sondas enviadas devido a condições de rede ou mecanismos de proteção, o *nmap* ajusta estes parâmetros considerando os valores limites definidos.

Com o objetivo de facilitar a configuração dos parâmetros acima referidos o *nmap* oferece *templates* de temporização baseados em níveis de agressividade dos *scans*. Existem seis níveis de agressividade (0-5) que se podem usar consoante o ambiente de operação da aplicação. A tabela 4.1 mostra os valores dos parâmetros de temporização dos vários *templates* de temporização [24].

Considerando o problema referido anteriormente foi necessário perceber que tipo de configuração é utilizada no *OpenVAS*. A configuração utilizada de momento era a *Full and very deep*. Esta configuração é a mais abrangente possível fazendo todos os testes de vulnerabilidades exceto os que possam causar DoS (*Denial of Service*) nos ativos a serem testados. A lista de portos utilizada era *All TCP and Nmap 5.51 top 100 UDP*. No *plugin nmap* o método de teste TCP definido é *TCP connect scan* e é utilizado o *template* de temporização Normal.

Para ter uma ideia do pior caso possível para o *scan* de descoberta de portos definimos as seguintes fórmulas (ignorando o parâmetro *Parallelization*):

$$\min(\textit{probes per second}) = \min(\textit{Max Scan delay}, \textit{Min Rate})$$

$$\max(\textit{duration}) = \frac{\textit{Max RTT Timeout} \times \textit{Número de portos} \times (\textit{Max Retries} + 1)}{\min(\textit{probes per second})}$$

	Nível	Initial RTT Timeout	Min RTT timeout	Max RTT Timeout	Max Parallelism
0	Paranoid	50 ms	100 ms	10 s	Serial
1	Sneaky	50 ms	100 ms	10 s	Serial
2	Polite	50 ms	100 ms	10 s	Serial
3	Normal	50 ms	100 ms	10 s	Parallel
4	Agressive	50 ms	100 ms	1250 ms	Parallel
5	Insane	50 ms	50 ms	300 ms	Parallel

	Nível	Scan Delay	Max Scan Delay	Max Retries
0	Paranoid	5 m	1 s	10
1	Sneaky	15 s	1 s	10
2	Polite	400 ms	1 s	10
3	Normal	0 s	1 s	10
4	Agressive	0 s	10 ms	6
5	Insane	0 s	5 ms	2

Tabela 4.1: Tabela de valores de temporização dos templates *nmap*

Por exemplo se ignorarmos o método de teste TCP e considerarmos os parâmetros utilizados na configuração *OpenVAS* referida anteriormente temos:

- *Max Retries*: 10.
- *Max Scan Delay*: 1 s (1 *probe per second*).
- *Max RTT Timeout*: 1250 ms.

$$10\ s \times 65535 \times (10+1) \div 1 = 7208850\ s \approx 83{,}4\ dias$$

É importante ter em consideração o pior caso pois, se este acontecer, os *vulnerability scanners* que o VAC utiliza vão estar preparados para devolver resultados em tempo útil, nomeadamente o *OpenVAS* que vai ser sempre utilizado como *vulnerability scanner* primário. Uma aproximação do pior caso pode ocorrer por exemplo se um ativo estiver protegido por uma *firewall* que faz cair todos os pacotes enviados exceto os que se destinem a portos abertos para este.

Para preparar o *OpenVAS* para estes casos foi necessário afinar os parâmetros da descoberta de portos de forma a limitar o tempo máximo de um *scan*.

Essa afinação começou pela redução dos valores máximos de temporização através da mudança do *template* de temporização de *Normal* para *Aggressive*. Com esta mudança são alterados 3 parâmetros importantes que definem o tempo máximo do processo:

- *Max Retries*: 6.
- *Max Scan Delay*: 10 ms (100 sondas por segundo).
- *Max RTT Timeout*: 1250 ms.

Recalculando o pior caso com estes parâmetros temos:

$1{,}25\ s \times 65535 \times (6+1) \div 100 = 5734{,}3\ s \approx 1{,}59\ horas$

Como referido anteriormente a afinação dos parâmetros de temporização pode implicar uma perda de precisão. Por exemplo se entre o *OpenVAS* e o ativo existir uma *firewall* que implemente uma política de limitação da taxa de pedidos (*rate limiting*) e se as afinações forçarem uma taxa de sondas enviadas (*probe rate*) que ultrapasse esse limite, a *firewall* vai começar a bloquear pedidos o que pode fazer com que não sejam descobertos portos abertos.

O ajuste explicado anteriormente reduz drasticamente o tempo máximo da descoberta de portos mas, considerando que este é apenas um dos testes iniciais de um *scan* de vulnerabilidades, é necessário reduzir ainda mais este valor. Para esse efeito foi decidido reduzir o número de portos a testar, sendo a seleção feita com base na popularidade e importância ao nível de segurança. Foram consideradas duas opções para esta escolha:

- ***Nmap 5.51 top 2000 TCP and top 100 UDP*** - Top 2000 de portos TCP e top 100 UDP na versão 5.51 do *Nmap*. Este top é definido no ficheiro *nmap-services* que é composto por uma lista com o nome do serviço, o protocolo de transporte, número do porto e um valor numérico que representa a regularidade com que o porto é encontrada aberto na Internet. Esta lista é compilada através de um conjunto de portos bem conhecido e uma investigação baseada em *scans* feitos na Internet, realizada pelo autor da ferramenta do Nmap[38][3][23].
- **Lista por omissão utilizada no *Nexpose*** - Conjunto de 1331 portos TCP e 109 portos UDP utilizados por omissão pelo *vulnerability scanner Nexpose*. A escolha de portos é uma seleção com base na experiência da equipa que cria o *Nexpose* bem como o registo de portos IANA e portos utilizados por trojans conhecidos[33][15].

A primeira listagem representa um cruzamento entre portos conhecidos e a regularidade destes encontrados abertos na Internet. A segunda listagem é um conjunto de portos específicos utilizado por um *vulnerability scanner* e representa a

experiência da equipa que desenvolve o *Nexpose* e algumas listas de conhecimento geral.

Sabendo que os *scans* a realizar serão feitos internamente e externamente foi decidido utilizar a lista por omissão utilizada pelo *Nexpose* visto esta ser uma escolha baseada em conhecimento sobre serviços e *malware* em geral, em vez da lista preterida, que apenas se baseia em exposição na Internet que muitas vezes não reflete a exposição interna em relação a um possível atacante presente na rede da empresa.

Esta listagem vai ser utilizada em *scans* realizados por ambas as tecnologias de *vulnerability scanning* de forma a existir uma lista padrão a ser utilizada pelo VAC.

A redução do número de portos diminui ainda mais a precisão da descoberta de portos visto que podem estar a correr aplicações vulneráveis ou maliciosas nos portos não incluídos por esta listagem. Para otimizar ainda mais a descoberta de portos foi escolhido o método de teste TCP SYN que apresenta os benefícios referidos anteriormente.

Visto que o contexto deste projeto se foca nas vulnerabilidades ao nível de serviços, sistemas e elementos de rede foi necessário analisar as configurações de *scan* do *OpenVAS* para perceber como testar apenas esse tipos de vulnerabilidades. Na organização das várias famílias de NVTs existe uma família em particular, *Web application abuses*, que lida com vulnerabilidades em aplicações *Web* que, não estando no contexto deste projeto, foi desativada.

4.1.2 *Nexpose*

O processo de *vulnerability scanning* no *Nexpose* é composto por três fases que ocorrem de forma ordenada:

- ***Asset discovery*** - Fase inicial onde é avaliado se um ativo está acessível na rede. Podem ser utilizados vários métodos como pedidos *echo* ICMP, ARP *pings* (para redes locais), pacotes TCP e UDP.

- ***Service discovery*** - Esta fase é iniciada apenas se o ativo estiver acessível. Consiste num processo de descoberta de portos TCP e UDP em que é utilizada a ferramenta *Nmap* para esse efeito. Por omissão, a configuração utilizada consiste num conjunto de 1331 portos TCP e 109 portos UDP, sendo que a escolha desses portos já foi explicada anteriormente. Em termos de temporização o *Nmap* tem os seguintes parâmetros mais relevantes:

  - *Max. retries*: 3.
  - *Max. RTT Timeout*: 3000ms.
  - *Max. Rate*: 15000.

De notar que nos parâmetros de temporização a configuração por omissão do *Nexpose* é muito mais restrita que a configuração por omissão do *OpenVAS*. O método de teste TCP SYN é utilizado de forma a melhorar a eficiência do processo.

- ***Vulnerability checking*** - Esta fase caracteriza-se pela execução dos testes de vulnerabilidades nos portos encontrados abertos na fase anterior. Os testes a executar são definidos pelo *scan template* escolhido.

O *scan template* utilizado no *Nexpose* foi o *Full audit without Web Spider* que nas duas primeiras fases utiliza as configurações por predefinição e na fase de *Vulnerability checking* executa todos os testes de vulnerabilidade possíveis exceto aqueles que se destinam a aplicações Web. Esta escolha permite a obter resultados em tempo útil pois as configurações estão afinadas por omissão, e não executa testes a vulnerabilidades Web visto estas não estarem no âmbito do projeto, o que permite também um ganho de tempo considerável na execução.

Ao contrário dos problemas encontrados com as configurações por omissão no *OpenVAS*, no *Nexpose* os *scans* executaram sempre com uma duração de tempo aceitável (sem os vulnerability checks associados a vulnerabilidades Web) o que já era expectável à partida visto ser uma ferramenta paga.

Como referido anteriormente, neste projeto foi preciso fazer uma escolha ponderada entre a precisão e a duração dos *scans* pelo que as afinações mencionadas anteriormente representam as configurações que devem ser utilizadas por omissão pelos *vulnerability scanners* *OpenVAS* e *Nexpose* no contexto do VAC.

No caso do *OpenVAS* a escolha de portos apresenta uma redução aceitável visto que engloba portos das aplicações mais utilizadas e de programas maliciosos conhecidos. Importante referir que esta lista deve ser revista periodicamente, em ambos os *scanners*, consoante a mudança de necessidades aplicacionais dos ativos a testar e o aparecimento de novos programas maliciosos.

As afinações de temporização reduzem bastante a duração dos *scans* e pressupõem a existência de uma ligação de boa qualidade entre ativos e os *vulnerability scanners*. Visto que existe um conhecimento prévio sobre as características de operação dos ativos, nomeadamente horas de maior intensidade operacional, os *scans* podem ser agendados para um período de menor atividade minimizando o impacto nos recursos dos ativos e na rede onde estes operam, bem como na precisão dos resultados do *scan*. Se as afinações de temporização interferirem com os mecanismos de segurança podemos utilizar as soluções vistas na secção 3.4 nos casos em que controlamos o ambiente de operação. Nos casos em que não podemos controlar os mecanismos de proteção devemos ajustar os parâmetros de configuração criando configurações especiais para estes.

O VAC deve suportar o uso de outras configurações não só para o caso mencionado anteriormente mas também para *scans* mais abrangentes que devem ser feitos aos ativos mais críticos a testar, caso de exista essa necessidade.

4.2 Configuração das plataformas de armazenamento de resultados

Segundo os requisitos, o VAC tem como finalidade armazenar resultados dos *scans* nas plataformas Hidra e *ArcSight* utilizadas pela DCY, pelo que foi necessário fazer uma preparação destas plataformas para receber dados provenientes do VAC.

Como mencionado anteriormente o Hidra é fundamentalmente composto por três tecnologias: *RabbitMQ*, *Elasticsearch* e *Kibana*.

Em primeiro lugar teve que ser criado um *virtual host* especifico no *RabbitMQ* para isolar a comunicação deste projeto de forma a não interferir com os sistemas em produção. Depois teve que ser definido nas configurações do VAC o nome dos índices onde os resultados são guardados no *Elasticsearch*. Inicialmente deve ser criado um *mapping* composto pelos campos e tipos de dados presentes nos resultados enviados pela aplicação. Finalmente os índices têm que ser adicionados no *Kibana* para podermos visualizar os dados na forma de eventos, tabelas, gráficos, etc.

O *ArcSight* recolhe os dados que utiliza através de conectores que estão instalados em várias máquinas que tipicamente recolhem informação através de *syslog*, consumo de ficheiros, etc. Estes conetores convertem o formato da informação recebida em eventos do *ArcSight* e envia-no para as máquinas que armazenam, processam e correlacionam estes.

Para a configuração do *ArcSight* foi pedido que se utilizasse os conectores já existentes de forma a minimizar o esforço por parte da equipa que gere o *ArcSight*. Sendo assim teve que se instalar um *SmartConnector* para o *Rapid 7 Nexpose* de forma a consumir relatórios no formato XML desta ferramenta.

Para integrar resultados do *OpenVAS* foi criado um conversor de relatórios do formato *OpenVAS* para *Nexpose*. A concretização deste será explicada em detalhe mais à frente. Um relatório ao ser convertido para o formato *Nexpose* é enviado para o *ArcSight* através do *SmartConnector Rapid 7 Nexpose*.

4.3 Avaliação do estado de segurança de ativos em termos de vulnerabilidades

A presença de vulnerabilidades num ativo tem um comportamento dinâmico ao longo do tempo pelo que é essencial realizar *scans* periodicamente para identificar as mudanças. Esse comportamento deve-se essencialmente aos seguintes fatores:

- Mudanças de estado no ativo que podem alterar o tipo e o número de vulnerabilidades a que este é exposto. Por exemplo instalação de novo software ou correções, abertura ou fecho de portos, atualização de software ou sistema operativo, etc.

- Descoberta de novas vulnerabilidades que são divulgadas sendo posteriormente criados *plugins* nas ferramentas de *vulnerability scanning* para descobrir estas.

- Condições de rede podem condicionar a descoberta de vulnerabilidades. Quando a rede está muito congestionada ou com falhas, um *vulnerability scanner* pode interpretar que uma vulnerabilidade não existe porque não recebeu um determinado conjunto de respostas que podem ter sido perdidas.

- Recursos dos ativos a testar. Um ativo que tenha poucos recursos pode ficar sobrecarregado com os pedidos gerados pelos testes enviados por um *vulnerability scanner*. Isto pode fazer com que o ativo alvo dos testes não consiga responder atempadamente aos pedidos, fique com serviços interrompidos ou interrompa o seu funcionamento, levando à possibilidade de algumas vulnerabilidades não serem descobertas.

Tipicamente, a estratégia passa pela realização de um *scan* para determinar o estado inicial de segurança em termos de vulnerabilidades de cada ativo (*baseline* em t_0). Esse *scan* normalmente gera um conjunto de resultados de descoberta de vulnerabilidades em que cada resultado tem uma pontuação CVSS associada.

A pontuação CVSS desses resultados permite perceber a severidade das vulnerabilidades encontradas e, consequentemente, permite avaliar a segurança do ativo em termos de vulnerabilidades. Tipicamente esta avaliação é feita através de uma função de agregação das pontuações CVSS de todas as vulnerabilidades de um ativo encontradas num *scan*. Esta avaliação, aliada a uma análise de criticidade para um conjunto ativos, permite definir prioridades para ações de correção e mitigação de vulnerabilidades nesses ativos. As pontuações CVSS também possibilitam a definição de prioridades de correção e mitigação de vulnerabilidades para um ativo. Por exemplo, uma vulnerabilidade mais grave deve ser corrigida ou mitigada antes de uma vulnerabilidade menos grave.

Como referido anteriormente, após a realização dos *scans* iniciais devem ser realizados, para cada ativo, *scans* de forma periódica ($t_{1...n}$) com o objetivo de detetar alterações no estado segurança em termos de vulnerabilidades.

Para cada *scan* existe um conjunto de resultados que deve servir para elaborar as avaliações e compilar relatórios para cada ativo. Esses relatórios são enviados aos responsáveis para que estes possam proceder à correção das vulnerabilidades encontradas.

No *scan* seguinte essas correções devem-se refletir nos resultados, ou seja, os resultados não vão incluir as vulnerabilidades corrigidas, o que contribui para a melhoria da avaliação de segurança de cada ativo na vertente de vulnerabilidades.

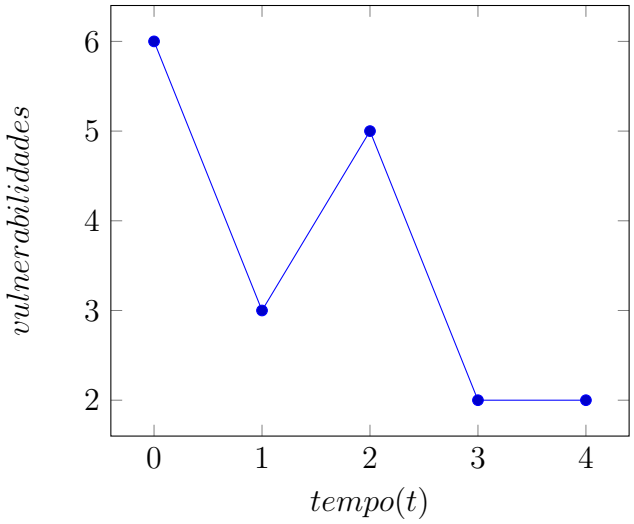

Figura 4.2: Evolução das vulnerabilidades num ativo ao longo do tempo

4.4 Definição da estrutura dos resultados

Um dos objetivos deste projeto é a abstração do VAC relativamente às tecnologias de *vulnerability scanning* que utiliza, logo é essencial que seja definido um formato canónico que represente resultados de descoberta de vulnerabilidades, de forma a que se possa comparar resultados de tecnologias diferentes e suportar a mudança de tecnologias. Por exemplo, se a DCY decidir mudar de tecnologia de *vulnerability scanning* os novos resultados podem ser facilmente incorporados e utilizados para efeitos de comparação com resultados anteriores de tecnologias de *vulnerability scanning* antigas.

Ambas as tecnologias de *vulnerability scanning* utilizadas possuem uma grande quantidade de informação pelo que foi preciso escolher um conjunto de atributos transversal e inequívoco na identificação de um resultado de descoberta de vulnerabilidade num ativo. Os campos escolhidos foram os seguintes:

- Nome da vulnerabilidade.
- Data de descoberta.
- Endereço IP.
- Protocolo de transporte.

- Número do porto.
- Resultado de teste.
- Tecnologia de *vulnerability scanning*.

O nome identifica a vulnerabilidade associada ao resultado, a data de descoberta o momento em que foi descoberta, o endereço IP o ativo onde esta foi encontrada. O protocolo de transporte e o número do porto identificam especificamente no ativo onde a vulnerabilidade encontrada. O resultado do teste está associado ao retorno da rotina utilizada para testar a vulnerabilidade. Este campo é útil pois podem existir dois testes que identifiquem a mesma vulnerabilidade e é necessário saber distingui-los e, também, informar o responsável pela segurança de um ativo sobre todos os detalhes relacionados que possam ajudar na correção da vulnerabilidade encontrada.

De forma a possibilitar a avaliação do estado de segurança de um ativo deve ser definida uma métrica associada aos resultados de descoberta de vulnerabilidades que permita definir um valor associado à sua severidade. Desta forma é possível definir uma função de agregação dos resultados de um ativo e atribuir uma pontuação de avaliação a este, bem como comparar pontuações entre *scans* e perceber a evolução do seu nível de segurança ao longo do tempo.

A métrica escolhida foi o CVSS visto esta ser um *standard* utilizado em todos os *vulnerability scanners* conhecidos, incluindo aqueles utilizados neste projeto. Neste *standard* o cálculo da pontuação de severidade de uma vulnerabilidade baseia-se na facilidade e impacto de exploração desta. O método de cálculo já foi explicado anteriormente na secção 2.7.

Portanto, para o VAC os campos importantes na definição de um resultado de descoberta de vulnerabilidade são os mencionados anteriormente. Não sendo essencial ao funcionamento do VAC, a informação adicional presente nos resultados dos *scans* é também guardada pois pode ser útil posteriormente no suporte à correção de uma vulnerabilidade. Um exemplo típico de informação adicional muito útil são as referências CVE (Secção 2.6). A figura 4.3 sumariza a escolha dos campos consoante o seu propósito.

No decurso do estudo e teste das duas ferramentas de *vulnerability scanning* percebeu-se que os mecanismos que lidam com resultados falso positivos são complexos e algo limitados, pelo que foi criado um mecanismo ao nível do VAC para lidar com falso positivos passando a complexidade da gestão destes para o lado da ferramenta.

Esta decisão facilita o desenvolvimento dos módulos das tecnologias de *vulnerability scanning*, assim como pode colmatar a falta desta funcionalidade numa ferramenta que possa ser utilizada no futuro.

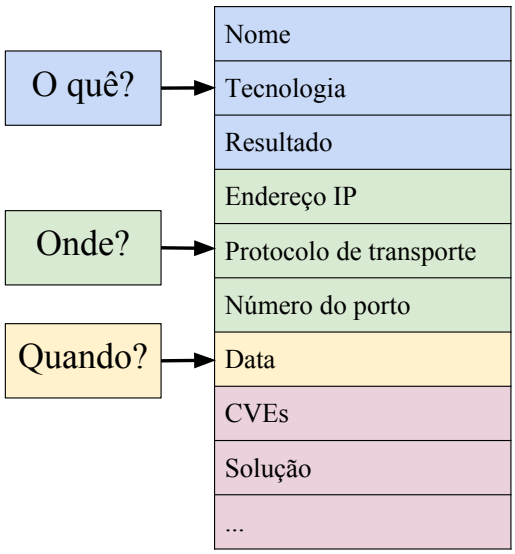

Figura 4.3: Campos de um resultado de descoberta de vulnerabilidade

O mecanismo criado baseia-se nos campos que definem a descoberta de uma vulnerabilidade, exceto a data de descoberta e a pontuação CVSS, sendo criada uma lista de exceções com base nesses campos. Quando um *scan* termina os seus resultados são comparados com essa lista antes de serem integrados nas plataformas de destino. Os resultados que igualarem elementos nessa lista não são considerados, ou seja, não são integrados nas plataformas de armazenamento de resultados.

4.5 Desenvolvimento do VAC

A arquitetura de software está estritamente relacionada com a Arquitetura do VAC pelo que a descrição da arquitetura vai ser dividida pelos componentes descritos na secção 3.3.

4.5.1 Decisões tecnológicas

A concretização da solução consistiu no desenvolvimento de uma aplicação que implementa a arquitetura definida e descrita no capítulo anterior. De forma a desenvolver a solução foi escolhida a linguagem de programação *Ruby*. Esta escolha teve como base um conjunto de características técnicas e restrições temporais específicas deste projeto.

O *Ruby* é uma linguagem que permite a utilização de vários paradigmas de programação, sendo que a solução foi concretizada segundo o paradigma de programação por objetos. Esta decisão foi tomada para que o desenvolvimento da aplicação seja dividido em vários componentes tornando, mais fácil modificar ou acrescentar fun-

cionalidades, e para conceder um comportamento modular às funcionalidades de gestão de *scanners* e integração de resultados. O facto de o *Ruby* ser uma linguagem reflexiva também contribuiu para a decisão, pois essa característica permite que sejam alterados módulos em tempo de execução não afetando a disponibilidade do VAC. O *Ruby* possui um conjunto rico de bibliotecas chamadas *gems* que permitiram a reutilização de código, agilizando o desenvolvimento do VAC permitindo manter o foco no desenvolvimento das respetivas funcionalidades.

4.5.2 Arquitetura de software

Considerando que o VAC foi desenvolvido segundo o paradigma de programação por objetos é importante descrever os vários objetos presentes na sua composição e a forma como estes se relacionam.

Inicialmente vai ser feita uma descrição de um conjunto de objetos genéricos que são essenciais na composição do VAC e são transversais aos vários componentes funcionais deste.

Classes genéricas:

Host

Representa os ativos a testar sendo um dos elementos principais do ambiente onde o sistema vai operar. Esta classe é composta por:

- ***address***: Endereço dos ativos que pode estar na forma de IP, *hostname* ou CIDR IP *range* no caso de representar um conjunto contíguo de ativos.
- ***properties***: Conjunto de propriedades que caracterizam o ativo como, por exemplo, a sua criticidade para a empresa, grupos a que pertença, sistema operativo, etc.

AvailablePeriod

Com o objetivo de reduzir o máximo possível a interferência dos *scans* nos ativos, conforme proposto nos requisitos, foi criado um objeto para limitar a janela temporal de execução dos *scans*. Este representa um período de disponibilidade diária, ou seja, o período do dia em que um *scan* pode executar. Este objeto é composto por:

- ***start_hour*** Horas de início do período diário.
- ***start_minute***: Minutos de início do período diário.
- ***end_hour***: Horas de fim do período diário.
- ***end_minute***: Minutos de fim do período diário.

HostGroup

Esta classe representa um agrupamento de ativos que serão alvos de *scans* ao mesmo tempo, tendo estes de ter disponibilidades compatíveis para a realização de *scans* de vulnerabilidades. No caso de *scans* com *templates* de configuração mais genéricos podem ser feitos grupos de ativos consoante um grupo lógico, por exemplo todos os servidores de uma determinada aplicação, ou podem ser feitos grupos consoante o tipo de serviços e aplicações que correm nos ativos, por exemplo grupo de servidores *Web*, e escolher um *template* mais focado num tipo de serviços e aplicações. Esta classe é representada por:

- **id**: identificador único que representa o grupo de ativos.
- **Lista de Host** : ativos que formam a composição do grupo.
- **AvailablePeriod**: período de disponibilidade comum dos ativos.

ScannerTemplate

Classe que descreve uma configuração de uma tecnologia de *vulnerability scanning* e é definida por:

- Designação da tecnologia de *vulnerability scanning* referida pela configuração de *scan*.
- *Hash* com valores que mapeiam uma determinada configuração de uma tecnologia de *vunerability scanning*.

A interpretação destas configurações vai ser da responsabilidade dos módulos que interagem com as tecnologias de *vulnerability scanning*.

Template

A configuração de ferramentas de *vulnerability scanning* é uma tarefa complexa que depende de um conjunto de fatores (secção 4.1) associados aos objetivos do *scan*, tendo em consideração as restrições temporais e do ambiente de operação. Esta tarefa depende de conhecimento sobre o funcionamento e experiência na utilização das ferramentas de *vulnerability scanning*. Para facilitar a utilização e abstrair o VAC das tecnologias de *vulnerability scanning*, na configuração de *scans* foi criada a classe Template que representa uma configuração de *scan* no VAC e é composta por:

- **id**: identificador único que representa o *template*.

- **Lista de ScannerTemplate**: mapeamento da configuração VAC para as configurações especificas das tecnologias.

Cada Template deverá ter um número de entradas na lista de ScannerTemplate igual ao número de tecnologias de *vulnerability scanning* utilizadas no momento. Esta lista deve ser atualizada à medida que as tecnologias utilizadas pelo VAC mudem. Esta classe tem como objetivo passar a complexidade de configuração dos *scans* para o administrador do VAC que é responsável por criar as configurações nos *scanners* das várias tecnologias de *vulnerability scanning* e mapear essas configurações em objetos do tipo Template. Quando os utilizadores criam um *scan* apenas escolhem uma configuração definida pela instância da classe Template. Isto permite que os utilizadores possam criar *scans* apenas com um entendimento de alto nível sobre as configurações.

Frequency

Para suportar a existência de *scans* que são executados com um período definido foi criada a classe Frequency. Esta define as várias frequências que um *scan* pode ter no VAC consistindo num método (*'next_period?'*) que fornecendo uma data e o tipo de frequência (diária, semanal, mensal, etc) calcula a data de execução do próximo *scan*.

A figura 4.4 demonstra um diagrama UML das classes descritas até agora e relações entre estas.

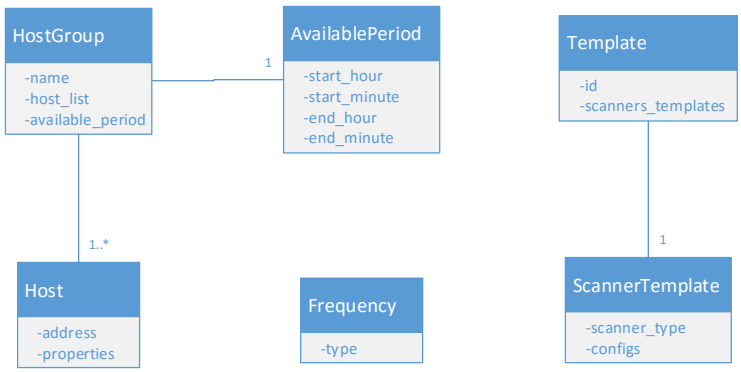

Figura 4.4: Esquema UML das classes genéricas do VAC

Componente Agendamento

Nos requisitos propostos, o VAC deve suportar a execução de *scans* de forma periódica para permitir realizar a avaliação contínua dos ativos, e também a execução

de *scans* de forma espontânea, para casos mais urgentes e específicos. Para cumprir esses requisitos foram criadas as classes PeriodicScan e SpontaneousScan.

PeriodicScan

Representa um *scan* do VAC que é realizado de forma periódica e é constituído por:

- **id**: identificador unívoco do *scan* periódico no VAC.
- **Hostgroup**: representação do conjunto de ativos alvo do *scan* periódico.
- **Template**: para determinar a configuração dos *vulnerability scanners* para o *scan*.
- Data e hora a que o *scan* deve iniciar.
- **Frequency**: representa a frequência do *scan* periódico.
- Data e hora da última vez que o *scan* iniciou (nulo antes do primeiro *scan*).
- ***Hash*** com opções adicionais para ajudar a caracterizar o *scan* periódico.

SpontaneousScan

Define os *scans* que são realizados espontaneamente e os seus atributos são:

- **id**: identificador unívoco do *scan* espontâneo no VAC.
- **Hostgroup**: representação do conjunto de ativos alvo do *scan* espontâneo.
- **Template**: designa a configuração dos *vulnerability scanners* para o *scan*.
- Tecnologia de *vulnerability scanning* utilizada.
- ***Hash*** com opções adicionais para ajudar a caracterizar o *scan* espontâneo .

Nesta classe é possível definir a tecnologia de *vulnerability scanning* pois este tipo de *scans* é utilizado em casos mais específicos onde, para realizar um *scan* urgente, pode ser necessário definir uma tecnologia em particular.

ScanScheduler

Classe que reúne os atributos da concretização do componente de agendamento do VAC.

- **ScannerManager**: objeto que representa o componente gestor de *scanners* e com o qual o componente de agendamento comunica para iniciar a realização de um *scan*.

- **Lista de PeriodicScan**: lista de *scans* periódicos definidos numa instância do VAC.

- **Lista de SpontaneousScan**: lista de *scans* espontâneos definidos numa instância do VAC.

- **Período de verificação de agendamento (segundos)**: intervalo de tempo entre duas verificações da listas de *scans*.

- **Tecnologia de *vulnerability scanning* primária**: *string* que, no contexto do VAC, represente a tecnologia de *vulnerability scanning* primária a utilizar nos *scans* periódicos.

A classe fornece métodos para criar e remover *scans* periódicos (PeriodicScan) e espontâneos (SpontaneousScan) às listas respetivas. Fornece também métodos para aceder a ambas as listas e para despoletar os *scans* no VAC, ou seja, enviá-los para o ScannerManager. Existe ainda um método que permite alterar a tecnologia de *vulnerability scanning* primária.

A figura 4.5 consiste num esquema UML das classes que compõe o componente de agendamento.

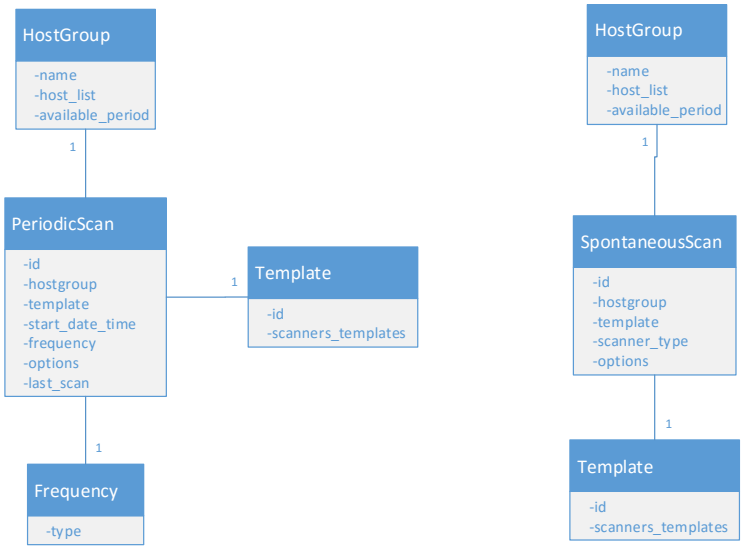

Figura 4.5: Esquema UML das classes que representam o componente de agendamento

Componente de Gestor de *Scanners*

Subcomponente *Scanners*

Para que o VAC seja o máximo possível independente das tecnologias de *vulnerability scanning* que utiliza foi necessário fazer uma generalização do conceito de *vulnerability scanner* e um mecanismo modular que permita que sejam alteradas as tecnologias de *vulnerability scanning* do VAC. Para isso, foi feita uma análise do comportamento dos *vulnerability scanners* utilizados neste projeto e um levantamento das ações necessárias para automatizar um *scan*. Isto resultou na criação de um objeto genérico que representa todos os *vulnerability scanners* que são e possam vir a ser integrados no VAC. Esse objeto reduz o *vulnerability scanner* a um conjunto de atributos e funções essenciais à realização de um *scan*, e é sobre estas funções que o VAC ordena a realização de *scans*. Depois existem módulos que fazem a ligação entre as funções do VAC e a tecnologia de *vulnerability scanning* a utilizar. Para concretizar este mecanismo foi utilizada uma aproximação do padrão de desenho de software *Factory Method* (Fig. 4.6 VulnerabilityScanner). Existe um objeto genérico VulnerabilityScanner que representa o conceito de um *vulnerability scanner* no contexto do VAC, concretiza o método que constrói os objetos que representam *vulnerability scanners* das tecnologias especificas e guarda referências das tecnologias de *vulnerability scanning* que podem ser criadas no VAC. Para cada tecnologia de *vulnerability scanning* que o VAC utiliza existe um objeto que concretiza as funções essenciais à realização de um *scan* nessa tecnologia.

De seguida são descritas as classes que concretizam o mecanismo mencionado anteriormente.

VulnerabilityScanner

Classe genérica que representa o conceito de *vulnerability scanner* no VAC.

- *subclasses*: atributo estático que guarda uma *Hash* com os objetos que representam tecnologias de *vulnerability scanning* utilizadas no VAC. Esta *Hash* utiliza como chave, para aceder ao objeto, uma referência (*String*) que identifica univocamente as tecnologias (Ex: *OpenVAS* - *openvas*, *Nexpose* - *nexpose*).

A classe fornece o método fábrica que constrói os objetos das tecnologias. Esse método tem como parâmetros a referência da tecnologia a utilizar, esta seleciona qual o objeto *scanner* que vai ser criado e atributos de configuração do *scanner*. Existe também um método que permite adicionar tecnologias ao VAC. Segundo a análise feita foi criado um conjunto de funções que permitem realizar as seguintes ações nos *scanners*:

- Iniciar, pausar e cancelar um *scan*.

- Consultar estado de um *scan* (iniciando, executando, terminado, etc.).
- Extrair e limpar resultados de um *scan*.

Como referido anteriormente, foram utilizadas as tecnologias *OpenVAS* e *Nexpose* pelo que foi criada uma classe para cada tecnologia.

OpenvasScanner

Como mencionado anteriormente (secção 2.8.1), é possível interagir com um *scanner OpenVAS* comunicando com o *OpenVAS Manager* através do protocolo OMP (*OpenVAS Management Protocol*). A comunicação é feita no porto 9390 do *Manager* utilizando o protocolo SSL de forma a garantir a segurança da comunicação, sendo esta composta por pedidos no formato XML definido pelo protocolo OMP. Para desenvolver a classe OpenvasScanner foi utilizada a *gem Ruby openvas-omp* presente no *GitHub* que, fornece um conjunto de métodos para as funções mais básicas, facilitando a comunicação ao realizar a tarefa trabalhosa de criar o pedido XML simplificando o código da classe[11]. Esta *gem* estava preparada para a versão anterior do protocolo OMP pelo que foi necessário fazer alguns ajustes para a *gem* funcionar com a versão OMP 6 utilizada pelo *OpenVAS* neste projeto.

Listagem de funções principais:

- ***initialize*** : cria um objeto do tipo OpenvasScanner com os atributos de configuração (*host*, porto, credenciais, etc.).
- ***start_scan*** : inicia um *scan* com um *id* fornecido.
- ***pause_scan*** : pausa um *scan* com um *id* fornecido.
- ***cancel_scan*** : cancela um *scan* com um *id* fornecido.
- ***resume_scan*** : inicia um *scan* com um *id* fornecido.
- ***scan_finished?*** : verifica se um *scan* terminou.
- ***scan_paused?*** : verifica se um *scan* está pausado.
- ***scan_stopped?*** : verifica se um *scan* está parado.
- ***scan_running?*** : verifica se um *scan* está em execução.
- ***generate_scan_results*** : extrai resultados de um *scan* e guarda-os num ficheiro.
- ***generate_incomplete_scan_results*** : extrai resultados de um *scan* incompleto e guarda-os num ficheiro.

- ***clean_scan_data*** : limpa os dados associados a um *scan*.
- ***active_scans?*** : retorna o número de *scans* ativos.
- ***scanner_type?*** : retorna a string associada à tecnologia de *vulnerability scanning*.
- ***id?*** : retorna o identificador do *scanner*.

Adicionalmente foram desenvolvidas outras funções de suporte menos importantes que não serão mencionadas.

NexposeScanner

A interação com um scanner *Nexpose* é feita através de uma *Web* API que este disponibiliza, sendo a comunicação feita utilizando o protocolo SSL. A API aceita pedidos no formato XML definido no manual. Para facilitar o desenvolvimento foi utilizada uma *gem Ruby* presente no *GitHub* e desenvolvida pela Rapid7[12]. Foram concretizadas as funções principais anteriormente expostas e algumas funções secundárias menos importantes.

As funções que comunicam com os *vulnerability scanners* estão protegidas com semáforos de forma a evitar que mais do que um pedido seja feito ao mesmo tempo a um *scanner*, o que poderia levar a comunicações incoerentes.

Subcomponente de acompanhamento

ScanInstance

Esta classe representa um *scan* ativo no VAC, ou seja, um *scan* que o módulo de agendamento mandou realizar. Os atributos desta classe são os dados necessários para a realização, acompanhamento e finalização de um *scan*. Identificam o *scan*, o seu estado e qual a instância de *vulnerability scanning* a que estão associados.

- **Identificador**: identifica univocamente o *scan* no VAC.
- **Hostgroup**: conjunto de ativos e as suas características pertinentes para a realização de *scans*.
- **Template**: designa a configuração dos *vulnerability scanners* para o *scan*.
- Identificador da instância de *vulnerability scanning* que realiza o *scan* (referência para VulnerabilityScanner).
- **ScannerManager**: objeto gestor de *scanners* a que o *scan* está associado.

- **Estado**: Estado de realização do *scan* (iniciar, executar, pausado, terminado).
- Data e hora a que o *scan* iniciou.
- Data e hora da última ordem para executar o *scan* (primeira execução ou última retoma).
- Data e hora do fim da realização do *scan*.
- Duração do *scan*.
- Referência que identifica univocamente o *scan* na instância de *vulnerability scanning*.
- Conjunto de opções adicionais para o *scan* (Ex: listagem de endereços de *e-mail* para notificação sobre o *scan*).

A classe fornece métodos que, de forma transparente, permitem iniciar, cancelar, pausar, resumir um *scan*, extrair resultados e limpar metadados associados ao *scan* no *vulnerability scanner*. Isto é possível pois esta classe utiliza apenas as funções essenciais que têm de ser concretizadas em todas as classes que herdam a classe VulnerabilityScanner. Existem ainda métodos de suporte para o envio de *e-mails* associados à atividade do *scan* e aos resultados obtidos como definido nos requisitos.

ScannerManager

Esta classe representa o módulo gestor de *scanners* sendo que os seus atributos definem os estados dos subcomponentes de acompanhamento e *scanners*. Esta classe é um ponto fulcral do VAC visto que representa a atividade principal deste (*scan* e obtenção de resultados).

De seguida são apresentados os atributos desta classe.

- **Lista de VulnerabilityScanner**: *vulnerability scanners* configurados no VAC.
- **Lista de ScanInstance**: *scans* ativos no VAC.
- **Lista de Template**: *templates* de *scans* no VAC.
- Caminho da diretoria que será o repositório dos relatórios recolhidos dos *vulnerability scanners*.
- **Período de verificação de acompanhamento (segundos)**: intervalo de tempo entre duas verificações da lista de *scans* ativos.
- Limite máximo de duração de atividade de um *scan*, conta o tempo do inicio ao fim incluindo o tempo de pausa.

- Limite máximo de duração da execução de um *scan*, apenas conta tempo em que o *scan* esteve a executar.

Esta classe expõe métodos que permitem:

- Criar, extrair resultados e apagar metadados de *scans* nos *vulnerability scanners*.
- Aceder à lista de *scans*.
- Adicionar e remover *vulnerability scanners* (VulnerabilityScanner).
- Aceder à lista de *vulnerability scanners*.
- Adicionar novas tecnologias de *vulnerability scanning* ao VAC (em tempo de execução).
- Adicionar e remover *templates* de *scan* (Template).
- Enviar *e-mails* com informação sobre atividade de *scans*.

A funcionalidade que permite carregar novas tecnologias de *vulnerability scanning* utiliza uma função que existe na classe VulnerabilityScanner para este efeito. Essa função aceita como argumento uma referência (*String*) que identifica univocamente a tecnologia no VAC e, essa função procura na pasta de instalação do projeto um ficheiro *Ruby* com o nome igual a essa referência. Se existir, carrega essa classe e executa-a. Essa classe por sua vez tem que herdar a classe VunerabilityScanner e executar o método que carrega a classe da tecnologia de *vulnerability scanning* na lista de subclasses da classe VulnerabilityScanner. A classe criada tem que ter um método construtor que suporte a chamada que é feita no método fábrica (*Factory Method*) e concretizar as funções essenciais que o VAC necessita para fazer a gestão de *scans*.

A figura 4.6 mostra como as classes apresentadas estão relacionadas na concretização do componente Gestor de *Scanners*.

Componente de Integracão

De forma a suportar a mudança das plataformas de armazenamento de resultados e novas tecnologias de *vulnerability scanning*, conforme descrito nos requisitos, foi necessário modularizar a integração de resultados.

Com isso em mente, foi decidido criar módulos para as várias tecnologias de *vulnerability scanning*, sendo cada módulo responsável por filtrar, fazer o tratamento dos resultados e enviá-los para todas as plataformas de armazenamento pretendidas e, se necessário, ordenar a realização de um *scan* de segunda opinião, ou seja, estes

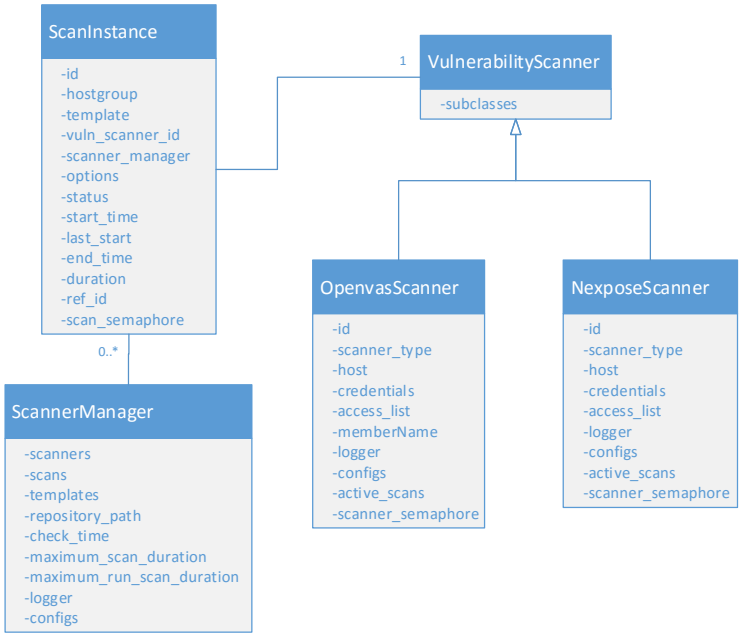

Figura 4.6: Esquema UML das classes que caracterizam o componente de gestão de *scanners*

módulos concretizam respetivamente os subcomponentes de Filtragem, Integração e Avaliação de uma forma faseada (Fig. 3.4).

A solução foi semelhante à utilizada nos *vulnerability scanners* e consistiu na generalização do conceito de um processador de resultados (Processor) e na existência de um módulo por cada tecnologia de *vulnerability scanning*. Cada módulo tem a capacidade interpretar resultados da tecnologia respetiva, descartar os resultados falso positivos utilizando uma base de dados num formato comum, transformar e enviar esses resultados para cada uma das plataformas de armazenamento pretendidas.

A classe genérica Processor tem um método fábrica (*Factory Method*) que constrói os objetos (módulos) que lidam com os resultados de cada tecnologia. Esses módulos precisam apenas de concretizar um método que deve implementar todas as operações descritas anteriormente sobre os resultados (*'send_results'*).

Eis uma descrição sucinta do desenho das classes utilizadas para a concretização apresentada anteriormente.

Processor

- ***processors***: *Hash* de objetos estática que guarda os processadores (objetos Processor) de dados para cada tecnologia.

Como referido anteriormente, esta classe concretiza o método que constrói os objetos para cada tipo de tecnologia de *vulnerability scanning*. Existem também

métodos para carregar novos objetos associados a novas tecnologias. Nos requisitos foi referida a importância desta modularidade funcionar em tempo de execução, sendo que, para adicionar um novo processador de dados ao VAC, apenas temos que chamar uma função com o nome da tecnologia para carregar o código *Ruby* desta. O ficheiro tem que ter o seguinte nome *'referência_tecnologia_processor.rb'* . Ainda existe um método que permite armazenar classes que herdem o tipo Processor na *Hash processors* sendo este método necessário para adicionar processadores de resultados de novas tecnologias ao VAC.

Foram concretizadas classes para as duas tecnologias utilizadas neste trabalho.

OpenvasProcessor

Nesta classe é concretizado o método '*send_results*' que é responsável por realizar todas as operações necessárias para a integração dos dados. Para cada plataforma existe um método que transforma, filtra e envia os resultados.

Os relatórios *OpenVAS* são extraídos no formato XML e para cada plataforma existem métodos responsáveis pela transformação e envio dos dados. O método *'send_results_arcsight'* faz a transformação do relatório XML do formato *OpenVAS* para o formato XML *Nexpose*. Isto foi concretizado desta forma para aproveitar o facto de o *ArcSight* integrar com o *Nexpose*, através do consumo de relatórios de *scans* de vulnerabilidades, sem a necessidade de desenvolvimentos por parte da equipa que gere o *ArcSight*. Os conectores do *ArcSight* consomem os ficheiros dos relatórios, pelo que neste momento é gerado o relatório XML e é colocado numa pasta para ser processado pelo conector.

Para enviar os dados para o Hidra o método '*send_results_hidra*' faz a transformação dos resultados presentes no relatório XML para registos simples, em que cada resultado é um registo. Após essa transformação os resultados são filtrados tendo em conta uma lista de falsos positivos presentes num registo da aplicação que é preenchido pelos utilizadores. Este registo contém uma lista de resultados falsos positivos que engloba todas as tecnologias de *vulnerability scanning* utilizadas pelo VAC, pelo que em cada processador são considerados apenas resultados da tecnologia que este processa.

NexposeProcessor

Assim como no OpenvasProcessor existe o método '*send_results*' que concretiza a integração dos dados nas plataformas de armazenamento de resultados.

O *Nexpose* possibilita a exportação dos relatórios em vários formatos pelo que foi escolhido o formato XML visto que apresentava mais informação que outros formatos.

Como referido anteriormente o *ArcSight* integra com o *Nexpose* consumindo relatórios XML de forma a integrar estes dados na sua base de conhecimento, neste caso o método '*send_results_arcsight*' apenas guarda os relatórios numa pasta sem nenhuma transformação.

O método '*send_results_hidra*', assim como no OpenvasProcessor faz a transformação e filtragem dos dados tendo em conta o formato de relatórios *Nexpose* e posteriormente envia estes para a plataforma Hidra.

Para os processadores de todas as tecnologias existe um método para avaliar a necessidade de ser executado um *scan* de segunda opinião com outra tecnologia de *vulnerability scanning*. Este método avalia a criticidade do ativo (*criticality*) para a empresa, sendo esta definida no processo de configuração de um *scan* no ativo. Se esse ativo tiver uma criticidade igual ou maior a 9, então o *scan* de segunda opinião é agendado. Este *scan* só é executado se a tecnologia do processador de resultados for definida como primária no VAC.

No caso da concretização realizada no âmbito deste trabalho os *scans* são realizados inicialmente com a tecnologia *OpenVAS*, se a condição para o *scan* de segunda opinião for válida, então é agendado um *scan* para os mesmos ativos com a tecnologia *Nexpose*.

ResultException

A classe ResultException representa um resultado que é considerado falso positivo e vai ser excepcionado em *scans* futuros. Este é composto por:

- Identificador único na aplicação para o falso positivo.
- Nome da vulnerabilidade excepcionada.
- O endereço do ativo onde esta foi encontrada bem como o protocolo de transporte e número do porto.
- O resultado da execução do *vulnerability check*.
- A tecnologia de *vulnerability scanning*.

Os atributos anteriores identificam univocamente a exceção na aplicação. Adicionalmente foram adicionados os seguintes atributos que são preenchidos pela pessoa responsável pela exceção:

- Justificação da exceção do resultado indicado.
- A pessoa que criou a exceção.

- O contacto dessa pessoa.

A classe tem métodos que permitem consultar todos os atributos que a representam. Adicionalmente tem um método devolve uma *String* que identifica univocamente a exceção, sendo esta formada pela concatenação dos seguintes atributos que representam o resultado excepcionado:

Nome + Endereço do ativo + Protocolo de transporte + Número do porto + Resultado da execução do vulnerability check+Tecnologia de vulnerability scanning

ResultsManager

Classe que representa o componente de Integração e engloba a transformação, filtragem e envio dos resultados para as plataformas de armazenamento de resultados. Esta é composta por:

- Pasta do repositório de resultados que contém os relatórios dos *scans* realizados pelos *vulnerability scanners*.
- Período de verificação de relatórios (segundos): intervalo de tempo entre duas verificações da pasta para encontrar relatórios novos.
- ScanScheduler que é utilizado para lançar *scans* de segunda opinião.
- Lista de exceções de resultados (falsos positivos).

Para configurar o componente de Integração de resultados a classe tem um conjunto de métodos que permitem:

- Adicionar e eliminar novos processadores de resultados, ou seja, processadores de tecnologias de *vulnerability scanning* novas, e consultar os processadores existentes na aplicação.
- Adicionar e remover exceções de resultados.
- Consultar a lista de exceções de resultados e verificar se uma determinada exceção existe.
- Modificar a pasta de onde irão ser consultados os relatórios dos *scans* de vulnerabilidades.
- Aceder ao ScanScheduler para despoletar *scans* de segunda opinião.

Tendo em conta que a quantidade de resultados processada por cada *scan* é normalmente elevada, teve que ser considerada uma maneira eficiente de verificar se um resultado está presente na lista de exceções. Consequentemente, esta foi concretizada utilizando uma *HashTable* devido à complexidade média de leitura ser $O(1)$, tornando as consultas muito rápidas. A chave da *Hash* é o identificador unívoco de uma exceção e o valor o objeto do tipo ResultException contendo os atributos que constituem a exceção.

Quando um relatório é processado, para cada resultado é construída uma *String* com base nos atributos deste, e é esta *String* que serve de chave para consultar a *HashTable* que representa a lista de exceções. Se uma exceção existir na lista o resultado é descartado.

No diagrama UML presente na figura 4.7 é clarificada a organização das classes mencionadas anteriormente.

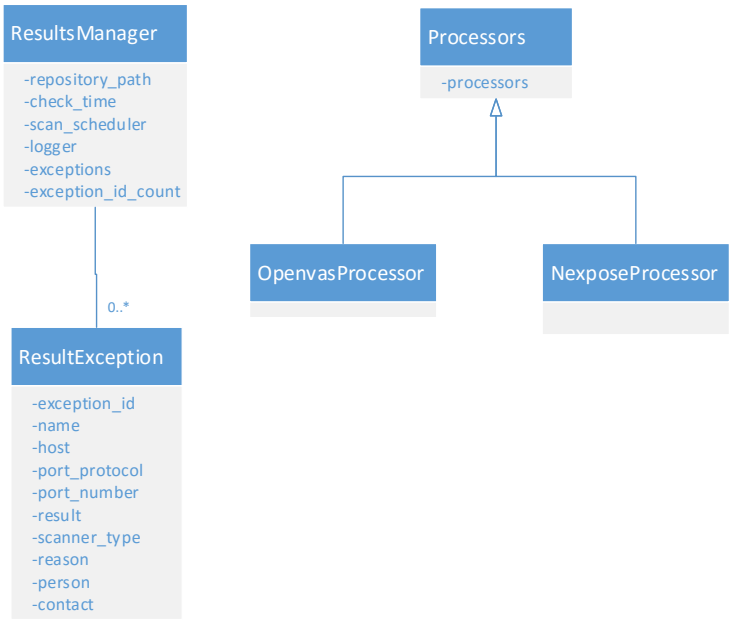

Figura 4.7: Esquema UML das classes que caracterizam o componente de integração de resultados

4.5.3 Componente de Interação

Este componente está concretizado sob a forma de um *web service* REST que tem como objetivo interagir com as várias funcionalidades da solução.

O *web service* foi realizado através de uma biblioteca e DSL (*domain-specific language*) chamada *Sinatra* que permite criar um *web service* em *Ruby* de forma fácil e rápida, o que possibilitou poupar tempo de desenvolvimento num compo-

nente periférico da solução[35]. Com o *Sinatra* é possível, com muito pouco esforço, configurar um *web service* sem a necessidade de instalar e configurar um servidor *web*. Estes foram fatores que levaram a esta escolha para a concretização do *web service*. O utilizador da aplicação interage com esta API através de pedidos HTTP direcionados para *routes* definidas e, em alguns casos, esses pedidos contém dados no formato XML.

A figura 4.8 apresenta uma lista de funções que a API disponibiliza.

Recurso	Método	Parâmetros	Resultados
POST	scan/periodic/		Adicionar um scan periódico
DELETE	scan/periodic/	:id	Eliminar um scan periódico
GET	scan/periodic/		Listar todos os scans periódicos
POST	scan/spontaneous/		Adicionar um scan espontâneo
DELETE	scan/spontaneous/	:id	Eliminar um scan espontâneo
GET	scan/spontaneous/		Listar todos os scans espontâneo
PUT	management/scanscheduler/checktime/		Alterar o periodo de verificação de agendamento do escalonador
GET	management/scanscheduler/checktime/		Consultar o periodo de verificação de agendamento do escalonador
PUT	management/scanscheduler/primaryscannertype/		Alterar a tecnologia de vulnerability scanning primária
GET	management/scanscheduler/primaryscannertype/		Consultar a tecnologia de vulnerability scanning primária
POST	management/scannermanager/scanner/		Adicionar um novo vulnerability scanner
DELETE	management/scannermanager/scanner/	:id	Eliminar um vulnerability scanner
GET	management/scannermanager/scanner/		Listar todos os vulnerability scanners
GET	management/scannermanager/scans/		Listar todos os scans
POST	management/scannermanager/template/		Adicionar um novo template de scan
DELETE	management/scannermanager/template/	:id	Eliminar um template de scan
GET	management/scannermanager/template/		Listar todos os templates de scan
POST	management/scannermanager/scannertype/		Adicionar uma nova tecnologia de vulnerability scanning
GET	management/scannermanager/scannertype/		Listar as tecnologias de vulnerability scanning configuradas
PUT	management/scannermanager/checktime/		Alterar o periodo de verificação de scans pelo gestor de scanners
GET	management/scannermanager/checktime/		Consultar o periodo de verificação de scans pelo gestor de scanners
PUT	management/scannermanager/repository/		Alterar pasta do repositório de resultados
GET	management/scannermanager/repository/		Consultar pasta do repositório de resultados
POST	management/resultsmanager/processor/		Adicionar um processador de resultados
DELETE	management/resultsmanager/processor/	:id	Eliminar um processador de resultados
GET	management/resultsmanager/processor/		Listar processadores de resultados
POST	management/resultsmanager/exception/		Adicionar uma exceção de um resultado
DELETE	management/resultsmanager/exception/	:id	Remover uma exceção de um resultado
GET	management/resultsmanager/exception/		Listar todas as exceções de um resultado
PUT	management/resultsmanager/checktime/		Alterar o periodo de verificação de relatórios
GET	management/resultsmanager/checktime/		Consultar o periodo de verificação de relatórios
PUT	management/resultsmanager/repository/		Alterar pasta do repositório de resultados
GET	management/resultsmanager/repository/		Consultar pasta do repositório de resultados

Figura 4.8: *Endpoints* da API REST da aplicação

A definição das *routes* e as funcionalidades a elas associadas estão presentes na classe ScanServer. Adicionalmente estão também algumas configurações do *Sinatra* que permitem que o servidor *web* lançado seja utilizado para servir a interface *web*.

A concretização da maioria das funções é composta pela interpretação do pedido XML e pela instanciação das várias classes necessárias com informação proveniente deste. Estas instâncias serão introduzidas nas estruturas de dados que representam o estado da aplicação. Como os pedidos podem conter muita informação a ser interpretada houve um cuidado especial com a *gem* ou biblioteca *Ruby* a utilizar e, após alguma investigação, concluiu-se que a *gem Nokogiri* é a mais eficiente a tratar de estruturas XML em Ruby[31].

De seguida serão mostrados alguns exemplos de pedidos XML feitos à API da aplicação e os processos que estes desencadeiam de forma a alterar o estado da

aplicação.

Criação de um *scan* periódico

```
<periodicscan>
   <id>teste</id>
   <hostgroup>
      <name>hostgroup_teste</name>
      <hosts>
         <host>
            <address>portal.teste.pt</address>
            <properties>
               <property>
                  <name>criticality</name>
                  <value>1</value>
               </property>
               <property>
                  <name>group</name>
                  <value>test_group</value>
               </property>
            </properties>
         </host>
      </hosts>
      <availableperiod>
         <starthour>12</starthour>
         <startminute>15</startminute>
         <endhour>23</endhour>
         <endminute>30</endminute>
      </availableperiod>
   </hostgroup>
   <template>Normal</template>
   <startdatetime>2019-02-02T00:30:29+00:00</startdatetime>
   <frequency>
      <type>daily</type>
   </frequency>
   <options>
      <option>
         <name>option1</name>
         <value>value1</value>
      </option>
   </options>
</periodicscan>
```

Figura 4.9: Estrutura XML de um pedido de criação de *scan* periódico

Quando este pedido (Fig. 4.9) é enviado para o serviço *web* da aplicação, a função ligada à *route* interpreta o pedido XML utilizando a *gem Nokogiri* e cria uma instância da classe Host com:

- **Endereço:** 'portal.teste.pt'.
- **Propriedades:** '*criticality*' com o valor '1' e '*group*' com o valor '*test_group*'.

Verifica se o *id* de *template* de configuração de *scan* fornecido existe na aplicação, se não existir a operação termina e devolve erro. Depois cria uma instância de Frequency com a string '*daily*' para definir o tipo desta e uma instância de AvailablePeriod com os atributos:

- ***start_hour:*** 12.
- ***start_minute:*** 15.
- ***end_hour:*** 23.
- ***end_minute:*** 30.

Depois cria uma instância da classe HostGroup com o *id* '*hostgroup_teste*' com o Host e AvailablePeriod previamente criados. Finalmente é criada uma instância de PeriodicScan com o *id* '*request1ps*', uma *Hash* com a opção adicional '*option1*' com o valor '*value1*', a data e hora de inicio do *scan* (*startdatetime*), o *id* do *template* de configuração a ser utilizado e as instâncias HostGroup e Frequency previamente criadas. A instância PeriodicScan criada é introduzida na lista de *scans* periódicos do ScanScheduler da aplicação.

Os outros pedidos que adicionam informação à aplicação seguem um método semelhante. Percorrem os elementos XML, que representam o nome das classes que vão construir ou o nome dos parâmetros a definir.

Consultar lista de *scans* periódicos

O pedido de consulta de *scans* periódicos (Fig. 4.10) devolve uma estrutura semelhante ao pedido de criação destes e, representa também a estrutura de objetos e parâmetros na lista de *scans* periódicos do ScanScheduler da aplicação. No momento em que este pedido foi feito a lista de *scans* periódicos tinha um *scan* configurado, sendo este composto por:

- **Identificador:** '*TestList3*'.
- ***HostGroup:*** grupo de ativos com o identificador '*TestList3Hostgroup*' que é constituído pelo ativo 'exemplo.telecom.pt' e caracterizado pela opção '*criticality*' a '2'. O período de disponibilidade diária para *scans* é das 00:00 às 23:59.
- ***Template:*** '*Refined*'.
- **Data de inicio do *scan*:** 2016-09-28 às 16:15.
- ***Frequency:*** frequência diária.
- ***Hash* de opções**: As chaves '*scan_activity_mail_list*' e '*scan_results_mail_list*' ambas com o valor 'user1@telecom.pt, user2@telecom.pt'.

Se existirem mais *scans* vão existir mais elementos '*periodicscan*' com a mesma estrutura demonstrada na figura 4.10.

```
<periodicscans>
    <periodicscan>
      <id>TestList3</id>
      <hostgroup>
        <name>TestList3Hostgroup</name>
        <hosts>
          <host>
            <address>exemplo.telecom.pt</address>
            <properties>
              <property>
                <name>criticality</name>
                <value>2</value>
              </property>
            </properties>
          </host>
        </hosts>
        <availableperiod>
          <starthour>0</starthour>
          <startminute>0</startminute>
          <endhour>23</endhour>
          <endminute>59</endminute>
        </availableperiod>
      </hostgroup>
      <template>Refined</template>
      <startdatetime>2016-09-28T16:15:00+01:00</startdatetime>
      <frequency>
        <type>daily</type>
      </frequency>
      <options>
        <option>
          <name>scan_activity_mail_list</name>
          <value>user1@telecom.pt, user2@telecom.pt</value>
        </option>
        <option>
          <name>scan_results_mail_list</name>
          <value>user1@telecom.pt, user2@telecom.pt</value>
        </option>
      </options>
      <lastscan>2016-10-31T16:16:27+00:00</lastscan>
    </periodicscan>
</periodicscans>
```

Figura 4.10: Estrutura XML da lista de *scans* periódicos do VAC

4.6 Interface *Web*

Conforme os requisitos, foi criada uma interface *Web* para que os utilizadores possam facilmente utilizar a aplicação. Esta interface foi criada recorrendo a um conjunto de técnicas de desenvolvimento chamado AJAX que permitem criar páginas *Web* que assincronamente enviam e recebem dados de um servidor. No caso deste projeto o servidor vai ser a API REST da aplicação. Isto vai permitir que a interface com o utilizador seja mais eficiente, pois apenas é necessário ser feito um pedido inicial para obter a página *Web* completa, alterações posteriores apenas necessitam de um subconjunto estritamente necessário de informação, que normalmente vem nos pedidos feitos ao servidor. A aplicação torna-se também mais resiliente a falhas do servidor, pois caso este esteja em baixo, a página continua ativa no cliente e facilmente retoma a operação quando o servidor voltar a funcionar.

Para facilitar o desenvolvimento da interface *Web* foi utilizada a *framework Bo-*

otstrap versão 3.3.7 que facilita a criação de páginas *Web* responsivas[4]. Foi utilizado o *template Starter* que é um *template* básico com uma página vazia com uma barra de navegação no topo. A partir desse *template* foi construida a interface que consiste numa barra de navegação com as secções:

- ***Current Scans:*** Lista de *scans* a decorrer.
- ***Scheduler:*** Consulta e definição da agenda de *scans*.
- ***Scanners:*** Consulta, adição ou remoção de *vulnerability scanners* da aplicação.
- ***Templates:*** Consulta, adição ou remoção de *templates* de configuração.
- ***Results:*** Consulta, adição ou remoção de processadores de resultados.
- ***Exceptions:*** Consulta, adição ou remoção de *excepções* de resultados.

Cada secção contém um conjunto de páginas e respetivos elementos HTML com *scripts Javascript* que compõem a interface. Adicionalmente existem *scripts Javascript* que tratam de fazer os pedidos assíncronos ao servidor.

Existem maioritariamente dois tipos de pedidos. A consulta de informação em que é feito um pedido de forma recorrente ao servidor para consultar informação e, é feita a leitura da resposta XML e a respetiva transformação para apresentar numa tabela (Ex: Tabela de *scans activos* - *Current Scans* 4.11). O outro tipo de pedido são as ações criadas pelo utilizador em que normalmente são preenchidos campos e depois o *script* consulta esses campos, constrói um pedido XML com essa informação e envia para o servidor (Ex: *Add Periodic Scan* em *Scheduler*).

A aplicação serve esta interface *Web* através do servidor *Web* criado na API REST pelo *Sinatra*. A figura 4.11 é um exemplo da interface Web que apresenta a listagem de *scans* que estão em execução pela aplicação. No topo da interface estão as restantes secções mencionadas na listagem anterior. Em geral cada secção trata de um tipo de objeto no contexto do VAC (agendamento, *scanner*, *template* de configuração, processador de resultados, exceção), e é possível criar novos objetos através da navegação de páginas onde são preenchidos os parâmetros necessários. Nessas secções também se podem consultar e eliminar esses objetos.

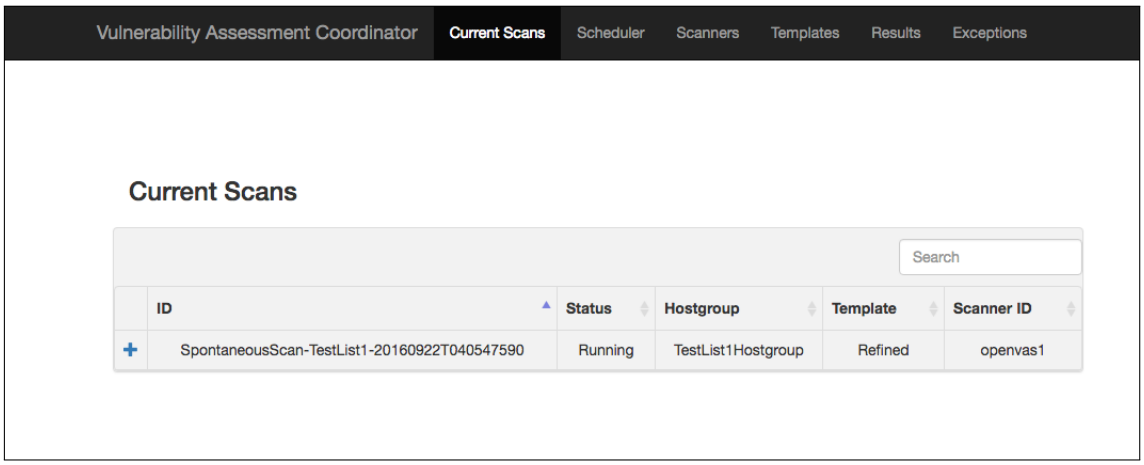

Figura 4.11: Página inicial com os *scans* em execução.

4.7 Tolerância a falhas da aplicação

Para tolerar falhas da aplicação esta armazena o seu estado em disco. O estado da aplicação é definido pelas instâncias das classes e pelos valores dos atributos destas.

Para cada classe da aplicação existe um método que permite exportar e importar o seu estado e, é com base nestes métodos que é permitido guardar e reconstruir o estado da aplicação em disco. Cada alteração que seja feita, nas estruturas que definam o estado da aplicação, é gravada no disco em tempo de execução. No contexto da aplicação, que opera num ambiente de grande escala, esse estado está em constante alteração e, tendo isto em consideração, foi dada especial atenção à forma como se guarda o estado no disco de forma a ser o mais eficiente e rápido possível. Foram investigadas algumas formas de serialização, com especial foco na eficiência e escrita para disco[2]. Foi escolhido o *MessagePack* que serializa os objetos de uma forma eficiente em formato binário[18]. Em *Ruby* está implementado na *gem msgpack-ruby*, sendo esta que foi utilizada para guardar os vários objetos que formam o estado da aplicação[10].

Caso a aplicação tenha uma falha, quando é recuperada, a aplicação carrega:

- Agendamento de *scans*.
- Estado de *scans* em execução.
- Exceções de resultados (falso positivos).
- Parâmetros de configuração da aplicação.

E, de seguida, retoma a operação com o estado que tinha antes de falhar. Esta funcionalidade é importante pois como a aplicação pode gerir um grande número de *scans* em simultâneo, em caso de falha, a aplicação recupera as referências para os

scans que estavam a decorrer nos *vulnerability scanners* na altura, evitando que o resultado desses *scans* seja desperdiçado.

4.8 *Log* da aplicação

A aplicação possui um *log* onde são registadas as principais operações e erros que possam ocorrer na aplicação, permitindo aos utilizadores e programadores perceber possíveis problemas que possam estar a ocorrer nesta que não sejam percetíveis na interface Web. Cada componente da aplicação regista no ficheiro de *log* todas as suas atividades e erros que ocorram.

A aplicação permite 3 níveis de *logging*:

- *ERROR:* Apenas regista erros.
- *INFO:* Regista erros e eventos da aplicação.
- *DEBUG:* Informação mais detalhada para a depuração de problemas (normalmente utilizado pelos programadores da aplicação).

O *log* da aplicação foi criado com recurso à biblioteca *Logger* do *Ruby*. A figura 4.12 mostra um excerto do *log* da aplicação VAC.

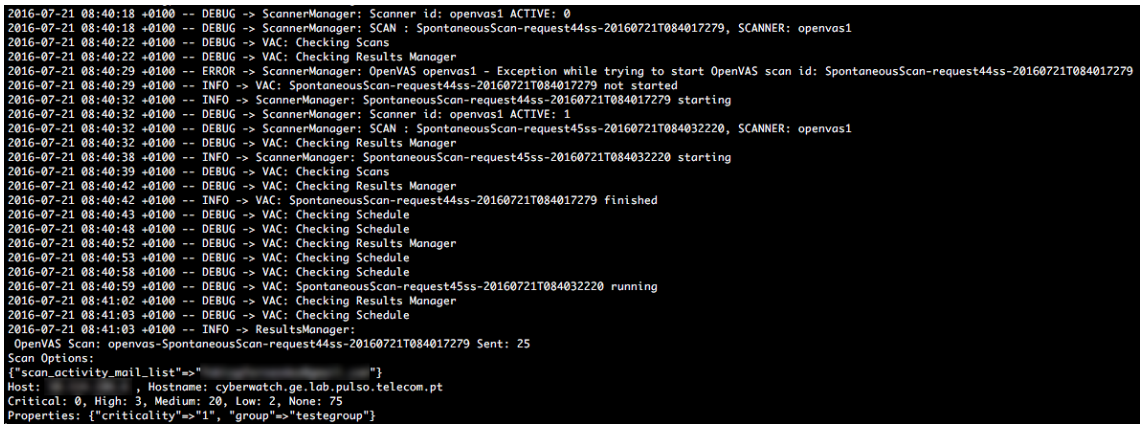

Figura 4.12: *Log* da aplicação VAC

4.9 Funcionamento e fluxo de execução da aplicação

Nesta secção é descrito o funcionamento geral da aplicação com o objetivo de tornar percetível a interação entre os vários componentes do sistema. É apresentado o exemplo da realização de um *scan* periódico, sendo este o caso mais comum da utilização da aplicação.

4.9.1 Inicialização da aplicação

Para iniciar a execução da aplicação tem que ser executado o *script* '*vac.rb*' que é responsável por inicializar as configurações iniciais, o estado e arrancar os componentes principais. Inicialmente o *script* lê um ficheiro no formato YAML ('*vac_config.yml*') que contém as configurações iniciais associadas aos vários componentes, como por exemplo o porto TCP que serve a API REST para a aplicação *Web* consultar, a diretoria onde são guardados os relatórios dos *vulnerability scanners*, o limite de tempo para um *scan* antes de ser interrompido forçosamente pela aplicação, etc. Depois verifica se existem ficheiros de estado da aplicação e recupera esse estado e, se não existirem, inicia a aplicação com um novo estado.

Com essas configurações e estados, a aplicação é então inicializada pelo lançamento das seguintes *threads* que representam os componentes da aplicação:

- ***Thread ScanServer:*** servidor *web* e serviço *web* (API REST).
- ***Thread ScanScheduler:*** gere o agendamento de *scans*.
- ***Thread ScannerManager:*** gestão de *vulnerability scanners* e *scans* em execução.
- ***Thread ResultsManager:*** envio de dados para as plataformas de armazenamento e tratamento de falso positivos.

A figura 4.13 apresenta um esquema que ajuda a perceber o fluxo de interações entre estas *threads*, tendo uma disposição semelhante à apresentada na arquitetura da aplicação (Fig. 3.1).

4.9.2 Agendamento de um *scan*

O utilizador interage com a interface *web* para criar um *scan* periódico com os parâmetros necessários, e esta, com base na informação preenchida pelo utilizador, cria um pedido XML e envia-o para o *endpoint* especifico da API REST ('*scan/periodic/*'). Ao receber este pedido, o serviço *web* da aplicação cria uma instância de PeriodicScan com a informação fornecida e envia a instância para a lista de *scans* periódicos da aplicação.

A *thread* ScanScheduler é composta por uma rotina que verifica periodicamente a lista de *scans* periódicos e espontâneos. Em cada verificação a lista de todos os *scans* é percorrida e os *scans* espontâneos são lançados imediatamente. No caso dos *scans* periódicos é calculado se está na altura de ser lançado segundo a seguinte condição:

$$Data\ Hora(agora) >= Data\ Hora(\textit{último scan}) + 1\ periodo(\textit{frequência scan})$$

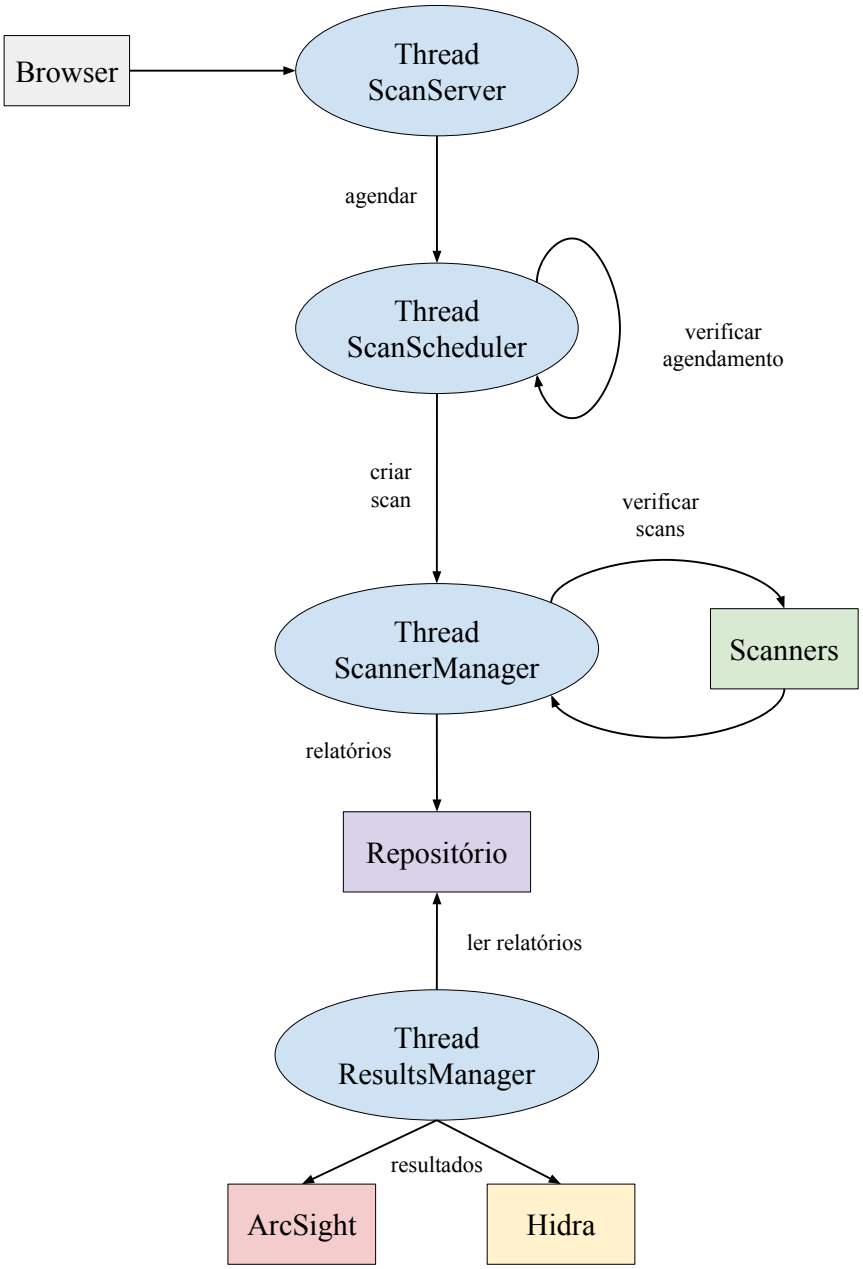

Figura 4.13: Funcionamento geral da aplicação

Para despoletar um *scan*, a *thread ScanScheduler* comunica com a *thread ScannerManager* (*'create_scan'*) e adiciona o *scan* à lista de *scans* a serem acompanhados pelo gestor de *scanners*. Normalmente os *scans* são lançados com a tecnologia de *vulnerability scanning* primária definida na classe ScanScheduler, apenas podem ser definidas tecnologias especificas em *scans* espontâneos que, normalmente são pedidos pelos utilizadores ou em *scans* de segunda opinião. Neste momento a tecnologia

primária é *OpenVAS*.

4.9.3 Lançamento de um *scan*

Quando é ordenado à instância ScannerManager a execução de um *scan* esta vai fazer uma consulta por todos os *vulnerability scanners* da tecnologia pedida e vai perceber qual tem o menor número de *scans* ativos a correr e associa esse *scanner* ao *scan* criado (ScanInstance). Este mecanismo permite que a carga dos *scans* seja distribuída pelos vários *scanners* configurados na aplicação.

A *thread* ScannerManager consiste também num ciclo periódico de verificação da lista de *scans* ativos (ScanInstance) no VAC. Para cada *scan* é verificado o seu estado, e consoante este, é despoletado um conjunto de ações que fazem o *scan* manter ou mudar de estado. A figura 4.14 descreve o fluxo dos estados percorridos até ao termino de um *scan*. Este pode começar no estado *New* ou *Paused* (caso o *scan* tenha sido pausado no *vulnerability scanner* por intervenção humana ou em caso de falha do *vulnerability scanner*).

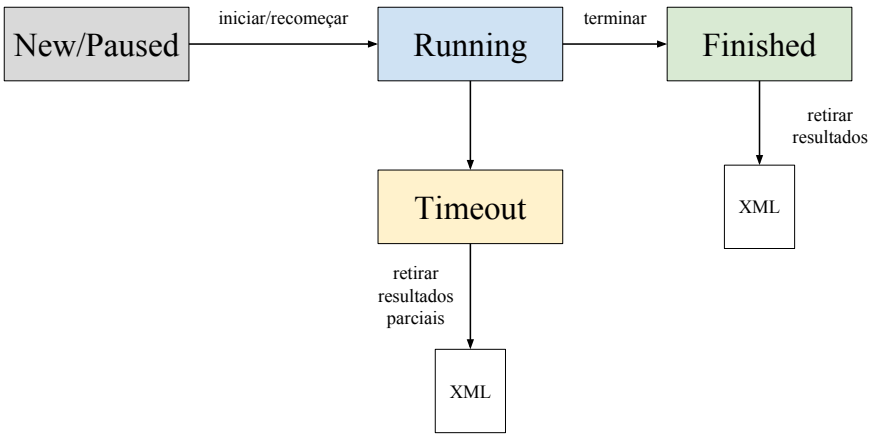

Figura 4.14: Esquema do fluxo de estados de um *scan*

4.9.4 Estado *New* e *Paused*

Quando um *scan* está no estado *New* o instante de tempo da verificação é considerado para verificar que o *scan* vai ser despoletado dentro da janela temporal de disponibilidade para o grupo de ativos. Para isso é consultada a instância AvailablePeriod relativa ao HostGroup da instância ScanInstance que representa o *scan* verificado. Se a condição for válida é procurado na lista de *vulnerability scanners* da instância ScannerManager o *vulnerability scanner* escolhido e é averiguado se este tem recursos suficientes para realizar o *scan*. Se tiver, o *scan* é lançado passando para o estado *Running*. O estado *Paused* é tratado de forma igual ao estado *New*,

mas em vez de o *scan* ser iniciado é recomeçado (*resume_scan*). A figura 4.15 mostra o fluxo de execução e a relação entre as principais classes envolvidas neste processo.

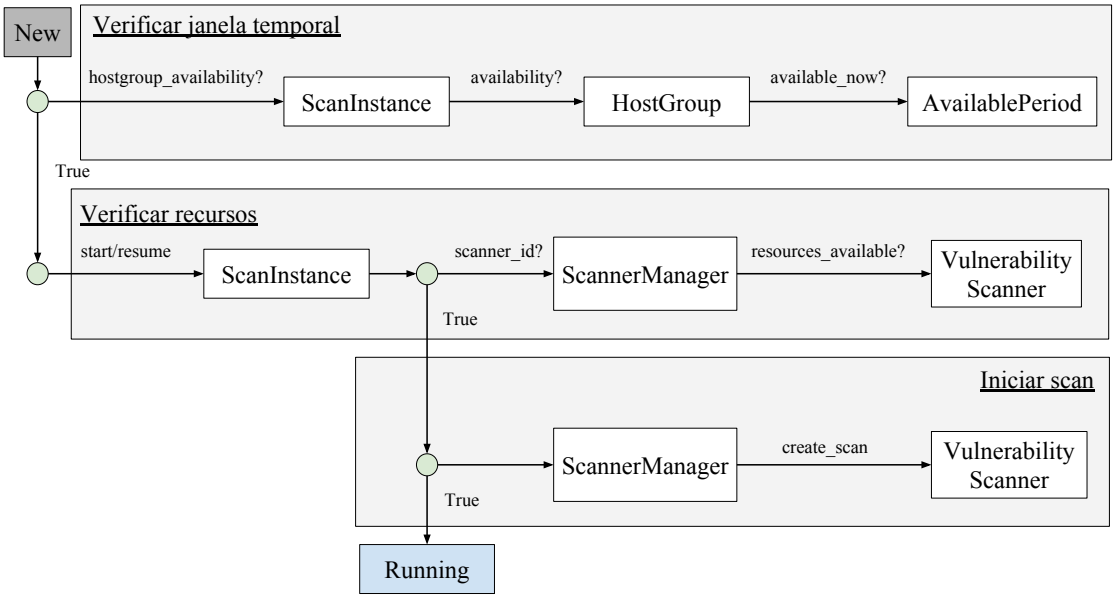

Figura 4.15: Fluxo de execução e classes envolvidas no inicio de execução de um *scan*

4.9.5 Estado *Running*

Se um *scan* se encontrar no estado *Running* tem que ser verificado se este não terminou no *vulnerability scanner* responsável pela sua execução. Para isso é executado o método *scan_finished?* na instância que representa o scanner (VulnerabilityScanner) associado ao *scan*. Se o resultado for verdadeiro o *scan* passa para o estado *Finished*, caso contrário vai ser verificado se o *scan* não excedeu a janela temporal de disponibilidade consultando a instância ScanInstance que representa o *scan*. Se não excedeu o *scan* continua com o estado *Running*, caso contrário o *scan* vai ser interrompido (*timeout*). Para isso a instância ScanInstance acede à instância de VulnerabilityScanner que, simboliza o *scanner* que está a executar o *scan*, e executa a função *timeout* que cancela o *scan*, passando este ao estado *Timeout*. A figura 4.16 resume a explicação anterior apresentando o fluxo de execução entre as várias classes envolvidas.

4.9.6 Estado *Finished* e *Timeout*

Caso o *scan* se encontre no estado *Finished*, então são retirados os resultados do *scan* e depois o estado deste é apagado no *vulnerability scanner* (Fig. 4.17), sendo posteriormente removido da lista de *scans* ativos.

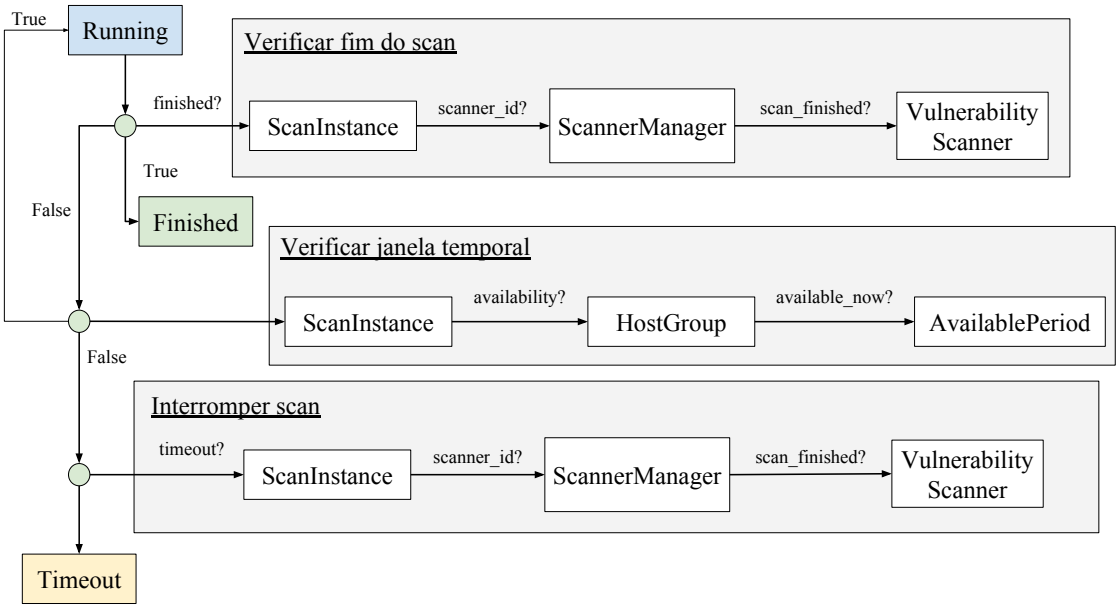

Figura 4.16: Fluxo de execução e classes envolvidas no acompanhamento de um *scan*

Com a finalidade de retirar os resultados, é executada a função *finish_scan* no ScannerManager, que por sua vez ordena a extração de resultados à instância ScanInstance (*generate_results*) que por sua vez procura a instância VulnerabilityScanner associada ao *scan* e ordena ao *vulnerability scanner* a extração dos resultados. No fim a instância ScanInstance é removida da lista de *scans*. O procedimento para o estado *Timeout* (Fig. 4.17) é semelhante ao *Finished* diferindo apenas no aspeto de que os resultados obtidos são incompletos. . Na figura 4.17 é resumida a interação entre as classes principais implicadas.

4.9.7 Processamento e integração de resultados

Quando os resultados são extraídos são guardados sob a forma de um ficheiro em que o nome segue o seguinte formato:

"*Tecnologia de vulnerability scanning − id do scan*"

Este formato vai permitir à *thread* ResultsManager perceber qual o módulo, associado à tecnologia de *vulnerability scanning*, a utilizar para fazer a transformação dos resultados. No fim dos resultados serem extraídos do *vulnerability scanner* é criado um ficheiro com o mesmo nome e a extensão '*.done*', sendo este a confirmação que o ficheiro com os resultados está pronto para ser lido.

A *thread* RusultsManager consiste numa rotina que periodicamente consulta a diretoria que representa o repositório de resultados e procura ficheiros com a ex-

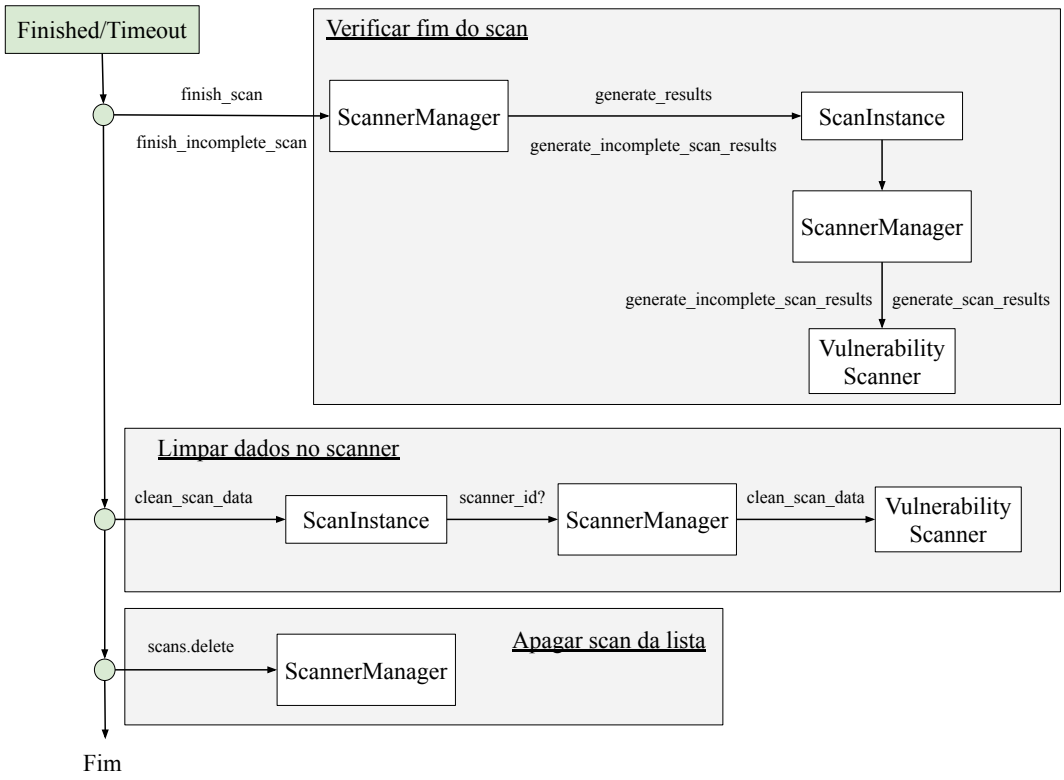

Figura 4.17: Fluxo de execução e classes envolvidas no término de um *scan*

tensão '*.done*'. Quando encontra, lê o prefixo do nome do ficheiro que representa a tecnologia de *vulnerability scanning* e utiliza a classe Processor adequada para lidar com essa tecnologia, executando o método '*send_results*' que é responsável pelo processamento, transformação, filtragem e envio de resultados para as plataformas de armazenamento. O método '*send_results*' consiste na execução de dois métodos. O método '*send_results_hidra*' que transforma os dados presentes no relatório XML para um formato não relacional, filtra falso positivos e envia os resultados na forma de eventos para a plataforma Hidra através da *gem* '*hidra*' criada pela equipa de desenvolvimento da DCY. Esta *gem* envia os dados para uma fila no *broker RabbitMQ*, sendo estes posteriormente consumidos por um processo que escreve os resultados num índice no *Elasticsearch*.

No final é verificado se a tecnologia de *vulnerability scanning* é a primária, utilizada normalmente pelo VAC. Se for, são avaliadas as propriedades dos ativos presentes no *scan* e, se for necessário, é executado um *scan* de segunda opinião com outra tecnologia. Neste momento é despoletado outro *scan* se a propriedade '*criticality*' for igual ou maior a 9.

De seguida é executado o método *'send_results_arcsight'* que trata do ficheiro com os resultados e envia-o para uma pasta predefinida de onde o conector *ArcSight* consumirá a informação que irá enriquecer a sua base de conhecimento. Quando

acaba a execução destes métodos os ficheiros com os relatórios são apagados.

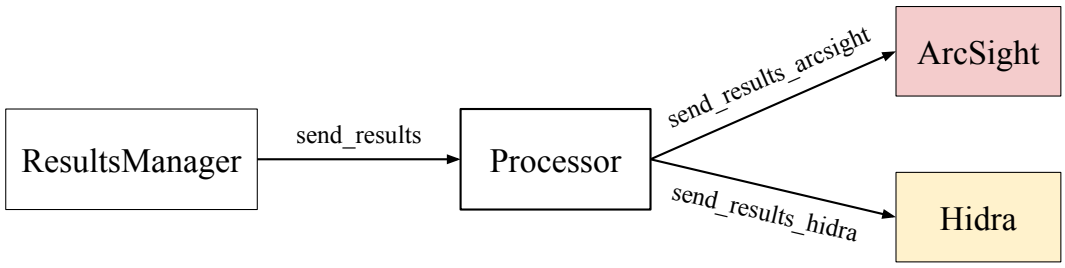

Figura 4.18: Fluxo de execução e classes envolvidas no processamento de resultados

4.10 Transformação de resultados

Para integrar os resultados nas plataformas de armazenamento presentes nos requisitos é preciso transformar os resultados para um formato que seja compatível com estas. Neste momento a aplicação suporta duas tecnologias, *OpenVAS* e *Nexpose*, e ambas possibilitam a exportação de relatórios no formato XML pelo que foi esse o formato escolhido em ambas, visto este formato ter uma maior quantidade de informação que os formatos preteridos. No ambiente de grande escala onde a aplicação vai operar, estes relatórios vão ter uma grande quantidade de resultados pelo que para fazer a leitura destes foi escolhida a *gem Nokogiri*, tendo a escolha já sido justificada anteriormente em 4.5.3. De seguida é explicado de forma sucinta o formato dos relatórios, são mencionados os segmentos principais de onde é retirada informação e é feita uma explicação da transformação efetuada para o formato a introduzir na plataforma Hidra.

4.10.1 *OpenVAS*

A figura 4.19 representa uma versão resumida e simplificada com o objetivo de auxiliar a explicação do processo de transformação dos resultados.

O formato de resultados do *OpenVAS* é bastante simples, para cada resultado encontrado num ativo num determinado porto é acrescentado um elemento *'result'* na lista de resultados (elemento *'results'*).

De uma forma simplificada o processo de transformação consiste em iterar nos vários elementos '*result*' e retirar a informação para o formato *Hash* (chave-valor) utilizado pelo Hidra. Os campos mais importantes retirados são:

- IP do ativo.
- Número e protocolo do porto que expõe a vulnerabilidade.

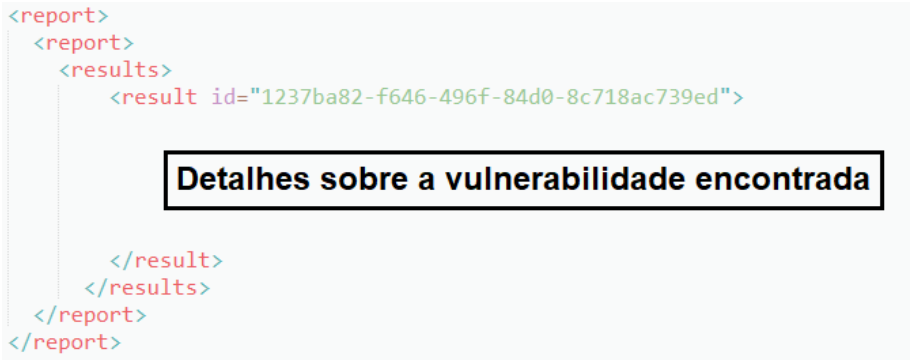

Figura 4.19: Estrutura XML simplificada de um relatório *OpenVAS*

- Nome da vulnerabilidade.
- Resultado da execução do teste.
- Pontuação CVSS.

Os restantes campos são específicos da tecnologia e não essenciais para a definição da vulnerabilidade. Exemplos: vetor CVSS, descrição, solução proposta, códigos CVE, lista de URL com informação, oid (*OpenVAS id*), etc.

4.10.2 *Nexpose*

A figura 4.20 mostra a estrutura geral de um relatório no formato XML *Nexpose*.

O relatório tem duas secções principais, os resultados dos ativos no elemento '*nodes*' e as definições de vulnerabilidades no elemento '*VulnerabilityDefinitions*'. As definições de vulnerabilidades contém informação detalhada sobre as vulnerabilidades que aparecem nos ativos do relatório e contém, por exemplo, o identificador da definição, o nome, a pontuação e o vetor CVSS, a pontuação PCI, descrição detalhada da vulnerabilidade, informação sobre *malware* e *exploits* que exploram esta vulnerabilidade, solução proposta, código CVE e outras referências. Como esta informação é extensa é guardada nesta lista de definições que depois podemos cruzar com a informação das vulnerabilidades encontradas nos ativos através do identificador (id).

O elemento '*nodes*' representa uma estrutura hierárquica que contem os resultados do *scan*. Cada elemento node representa um ativo e dentro deste existem os elementos '*tests*' e '*endpoints*'. A primeira representa os resultados que não estão associados a nenhum porto e o segundo os resultados que têm essa associação. Cada elemento '*endpoint*' tem um número de porto, protocolo associado e um conjunto de serviços, cada serviço tem o conjunto de resultados associados a este.

O processo que realiza a transformação dos dados começa por construir uma *Hash* com as definições das vulnerabilidades ao percorrer a lista de elementos '*vulnera-*

Figura 4.20: Estrutura XML simplificada de um relatório *Nexpose*

bility' dentro de '*VulnerabilityDefinitions*', onde a chave é o identificador (atributo *id*) da definição e o valor os detalhes da vulnerabilidade. A segunda fase consistem em percorrer a hierarquia de elementos '*node*' e primeiro transformar as vulnerabilidades sem porto associado (elemento '*tests*') e depois percorrer a lista de elementos '*endpoint*', os serviços (elementos '*service*'), os resultados encontrados (elementos '*test*') e transformar estes também para um formato *Hash* que contem a seguinte informação:

- Identificador da vulnerabilidade.
- IP do ativo.
- Número e protocolo do porto que expõe a vulnerabilidade.
- Resultado da execução do teste.

Depois é consultada a *Hash* com os detalhes das vulnerabilidades utilizando o identificador da vulnerabilidade e é retirada mais informação para construir o evento que vai representar um resultado. No fim os eventos são filtrados de resultados falso positivos que possam existir e depois são enviados para a plataforma Hidra.

4.11 Conversão do formato *OpenVAS* para *Nexpose*

Um dos requisitos é a integração dos resultados dos *scans* na plataforma *ArcSight*. Esta apenas suporta relatórios no formato *Nexpose*. Para poupar recursos à equipa que gere o *ArcSight* foi concretizada uma funcionalidade no VAC que, quando obtém resultados no formato *OpenVAS*, faz a conversão destes para o formato *Nexpose* com o máximo de precisão possível. A ideia principal é passar de um formato simples de registos do *OpenVAS* para a estrutura hierárquica do *Nexpose*.

Na primeira fase é construido o esqueleto do relatório XML, são percorridos os resultados do relatório *OpenVAS* para descobrir que IPs e portos destes contém vulnerabilidades e é criada a estrutura hierárquica de '*nodes*' (ativos) e dentro destes os '*endpoints*' (portos) conforme o formato *Nexpose* explicado anteriormente. Sendo que o *OpenVAS* não apresenta o nome do serviço como o *Nexpose*, todos os resultados ficam sobre um serviço criado concatenando o número do porto e o protocolo.

Na segunda fase são percorridos todos os IPs e para cada um são consultados todos os resultados. Para cada resultado são construidos o elemento '*test*' com o *id* do elemento '*result*' do relatório *OpenVAS* e '*vulnerability*' com os detalhes da vulnerabilidade. O elemento '*vulnerability*' é adicionado à lista de definições de vulnerabilidades ('*VulnerabilityDefinitions*') e se o valor do porto TCP for um número então o elemento criado vai ser colocado sobre o elemento '*endpoint*' que represente esse porto, senão vai para a lista de resultados sem porto associado (elemento *'tests'*).

Capítulo 5

Avaliação

Neste capítulo é demonstrado que os requisitos propostos para o VAC foram cumpridos (Secção 3.2). Foi realizado um conjunto extensivo de testes, 13 no total, com o objetivo de testar a funcionalidade básica de orquestração e agendamento de *scans* e, o consequente armazenamento de resultados proveniente destes (testes 1, 2, 3 e 9), a escalabilidade e desempenho da aplicação, dos seus componentes e das suas configurações (testes 4, 7 e 10), extensibilidade da aplicação a novas tecnologias de *vulnerability scanning* (teste 5), a funcionalidade de *scan* de segunda opinião (teste 6), filtragem de resultados falso positivos (teste 8) e, finalmente, a resiliência da aplicação face a falhas nos vários componentes (teste 11, 12 e 13).

A metodologia para a realização dos vários testes é muito semelhante e começa tipicamente com a configuração da interface da aplicação para executar uma determinada tarefa, consoante o objetivo do teste, pelo que são mostradas imagens da interface sempre que pertinente. Para mostrar o resultado de uma determinada tarefa por vezes foi necessário mostrar também imagens da aplicação e, em alguns casos, são mostrados excertos do *log* que ajudam a detalhar as ações realizadas pela aplicação. Noutros casos são mostradas imagens da interface dos *vulnerability scanners* para demonstrar a interação da aplicação com estas. Em alguns testes foi necessário manipular configurações nos ativos para facilitar a demonstração (por exemplo, expor serviços vulneráveis, desligar o ativo, etc.). O objetivo principal do funcionamento do VAC é obter resultados de *scans*, pelo que na maioria dos testes são apresentadas imagens das interfaces das plataformas de armazenamento de resultados (Hidra e ArcSight) com os resultados provenientes de um *scan*. Para obter os resultados armazenados no Hidra, foram feitas pesquisas na interface *Kibana* pelo nome do *scan* gerado na aplicação, no *ArcSight* as pesquisas foram feitas pelo nome do relatório gerado pela aplicação, que tem o nome do *scan*.

5.1 Teste 1 - *Scan* espontâneo e integração Hidra e *ArcSight*

O primeiro teste consiste no lançamento de um *scan* espontâneo, sendo criado pelo utilizador, sobre um conjunto de 5 ativos. A janela de disponibilidade é das 00h:00m às 23h:59m, o *template* de configuração utilizado foi *'Refined'* (resultado da afinação de *scans OpenVAS* na secção 4.1) e a tecnologia de *vulnerability scanning* o *OpenVAS*.

Para configurar o *scan* foi utilizada a interface gráfica seguindo um conjunto de passos para a configuração e lançamento do *scan*. Foi escolhido o separador de agendamento de *scans* e pressionado o botão '*Add Spontaneous scan*' (Fig. 5.1).

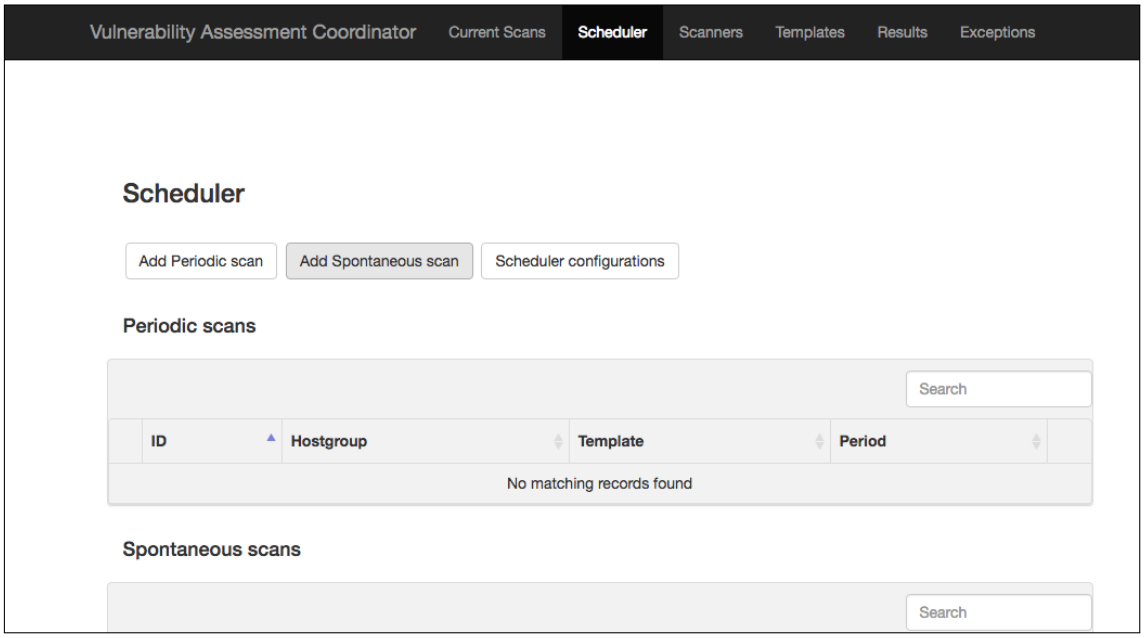

Figura 5.1: Secção *Scheduler*

A seguir foram preenchidas as configurações do *scan* espontâneo (Fig. 5.2, 5.3, 5.4).

Depois do *scan* ser criado fica na lista de *scans* espontâneos esperando que o módulo de agendamento faça a leitura da lista e inicie a sua execução imediatamente (Fig. 5.5).

O *scan* é lançado pelo gestor de *scanners* e passa a ser mostrado no separador *Current Scans* (*SpontaneousScan-TestList1-20160922T040547590* na figura 5.6).

Quando passa para o estado *Running* significa que a sua execução foi iniciada no *vulnerability scanner*, neste caso numa instância de *OpenVAS* (Fig. 5.7).

No fim do *scan* foi gerado um relatório que foi processado e enviado para as fontes Hidra e *ArcSight*. Como podemos observar na interface *Web* do *Kibana* (Fig. 5.8) os resultados do *scan* '*SpontaneousScan-TestList1-20160922T040547590*'

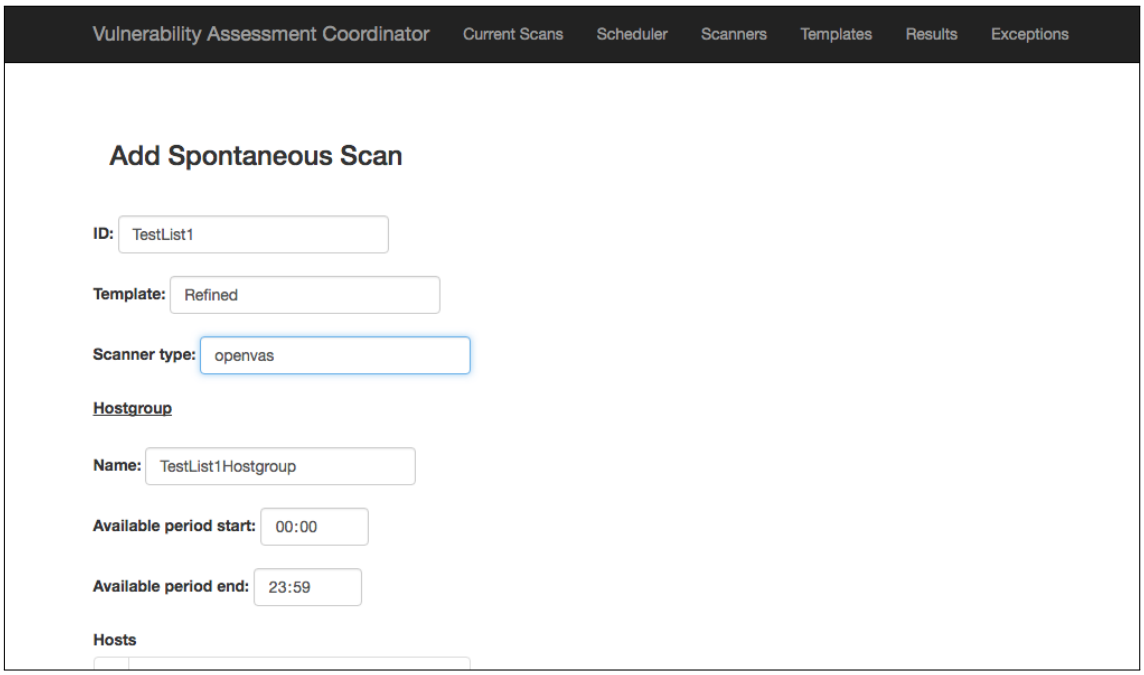

Figura 5.2: Preencher *Spontaneous scan* 1

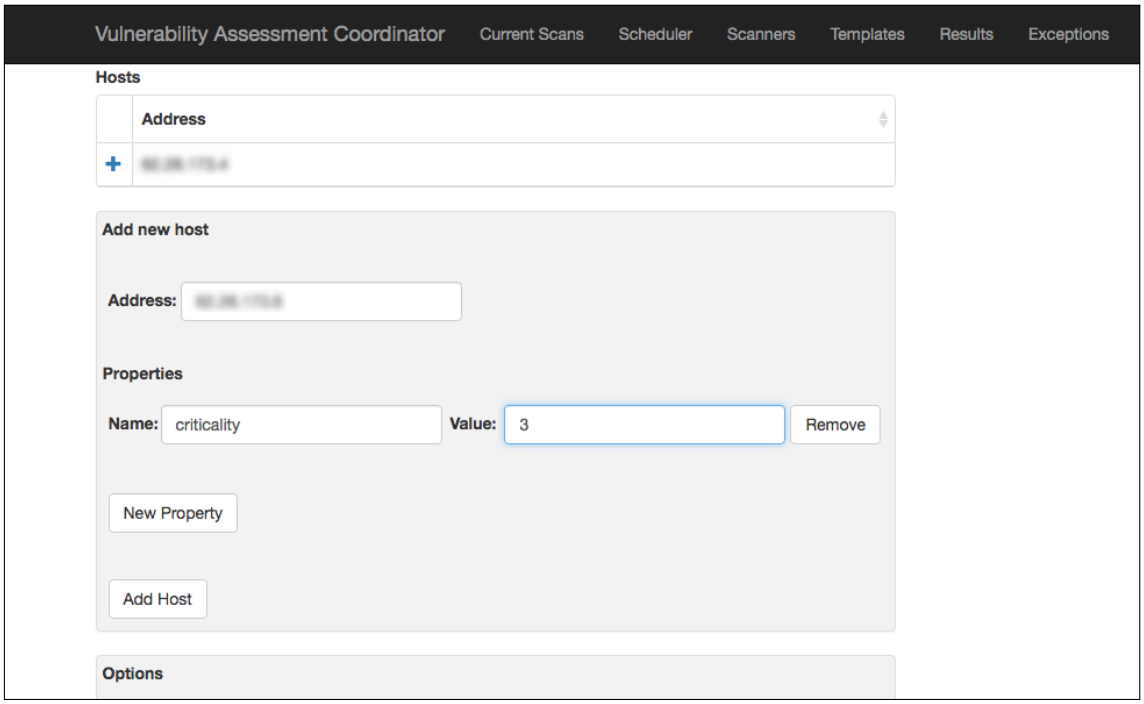

Figura 5.3: Preencher *Spontaneous scan* 2

foram introduzidos no *Elasticsearch*. Os resultados do formato *OpenVAS* foram transformados para o formato *Nexpose* e foram integrados na plataforma *ArcSight* (Fig. 5.9).

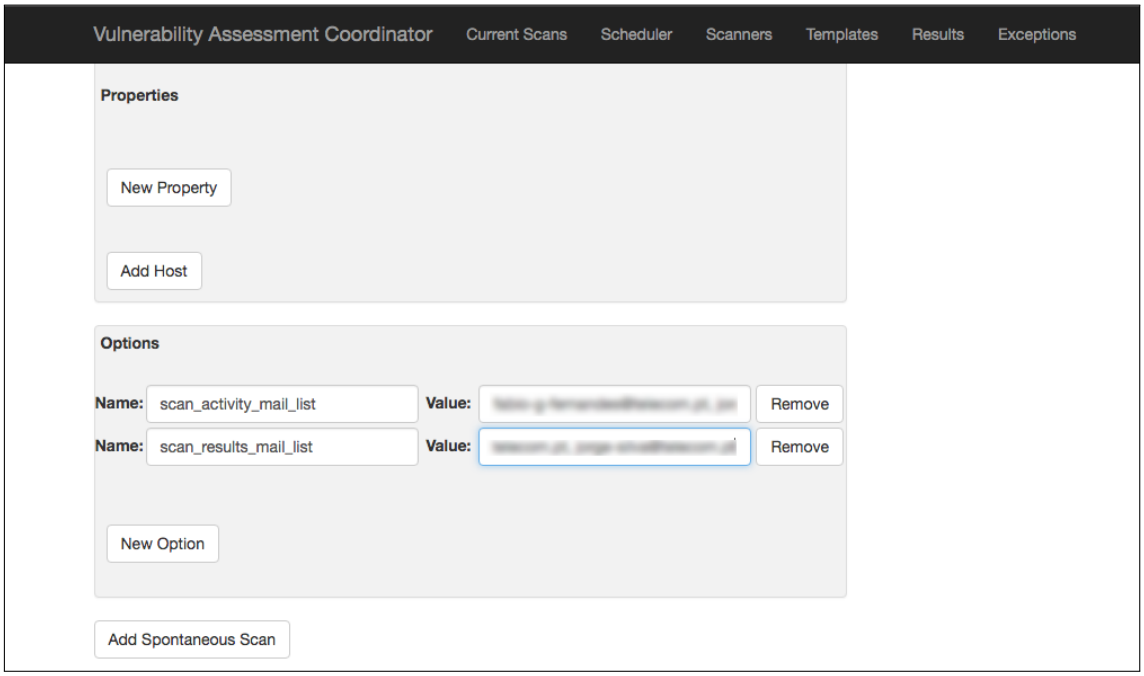

Figura 5.4: Preencher *Spontaneous scan* 3

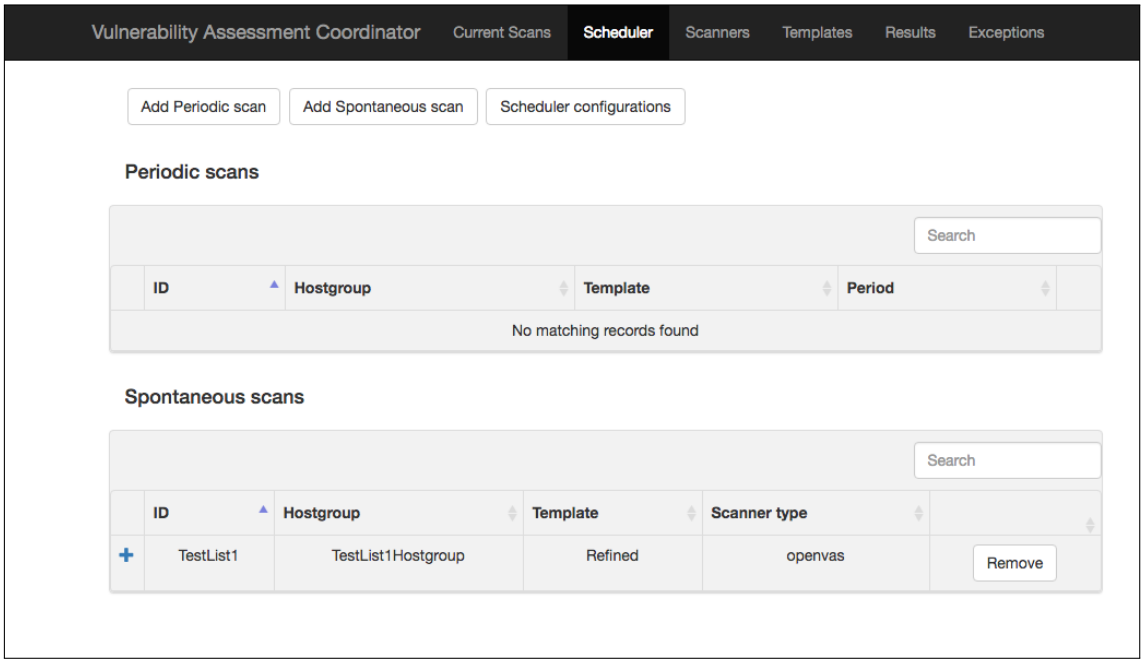

Figura 5.5: Secção *Scheduler* - *Scans* agendado

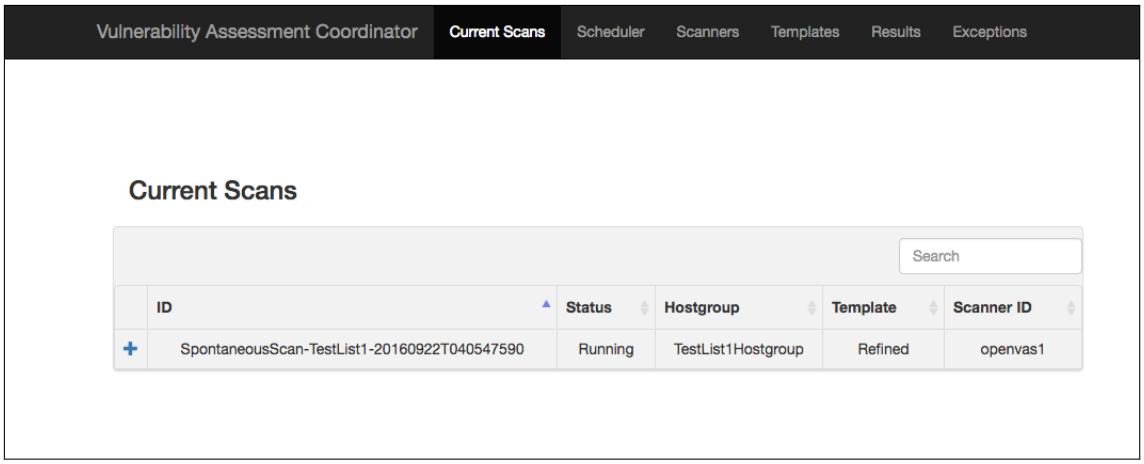

Figura 5.6: Secção *Current Scans* - *Scans* ativos

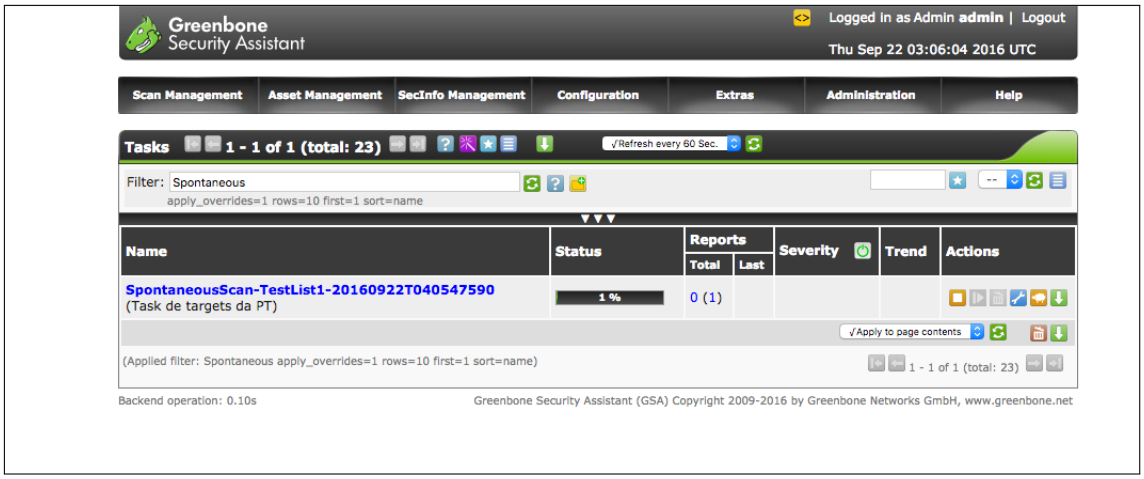

Figura 5.7: *Scan* em execução no *OpenVAS*

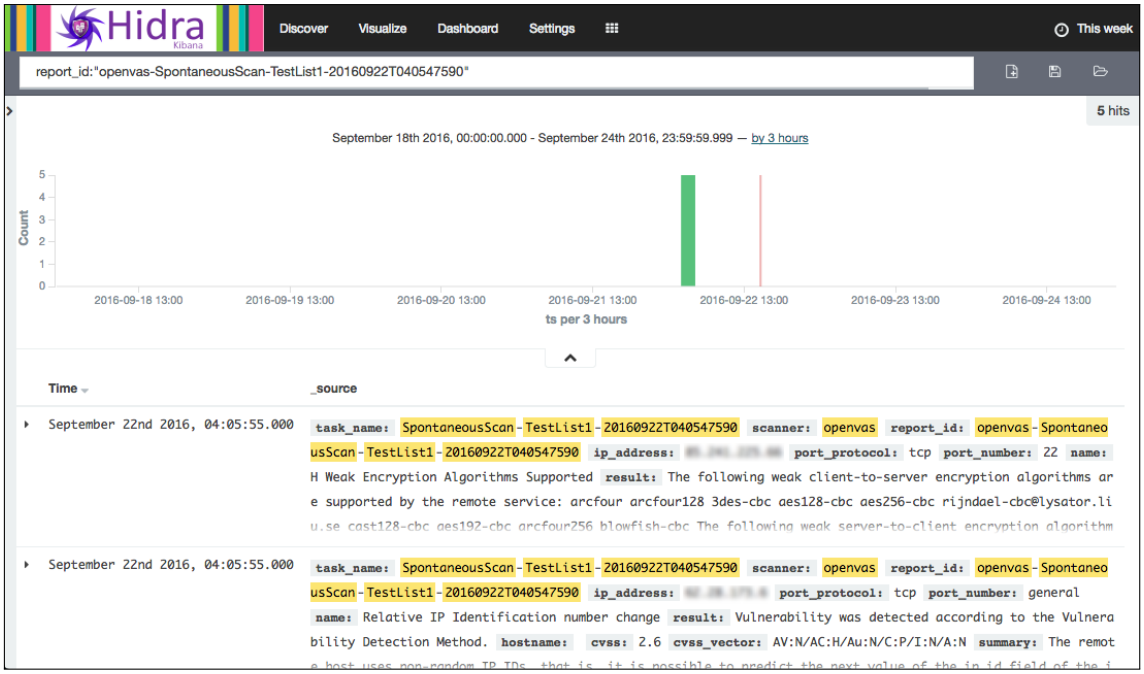

Figura 5.8: Resultados do *scan* do teste 1 no *Kibana*

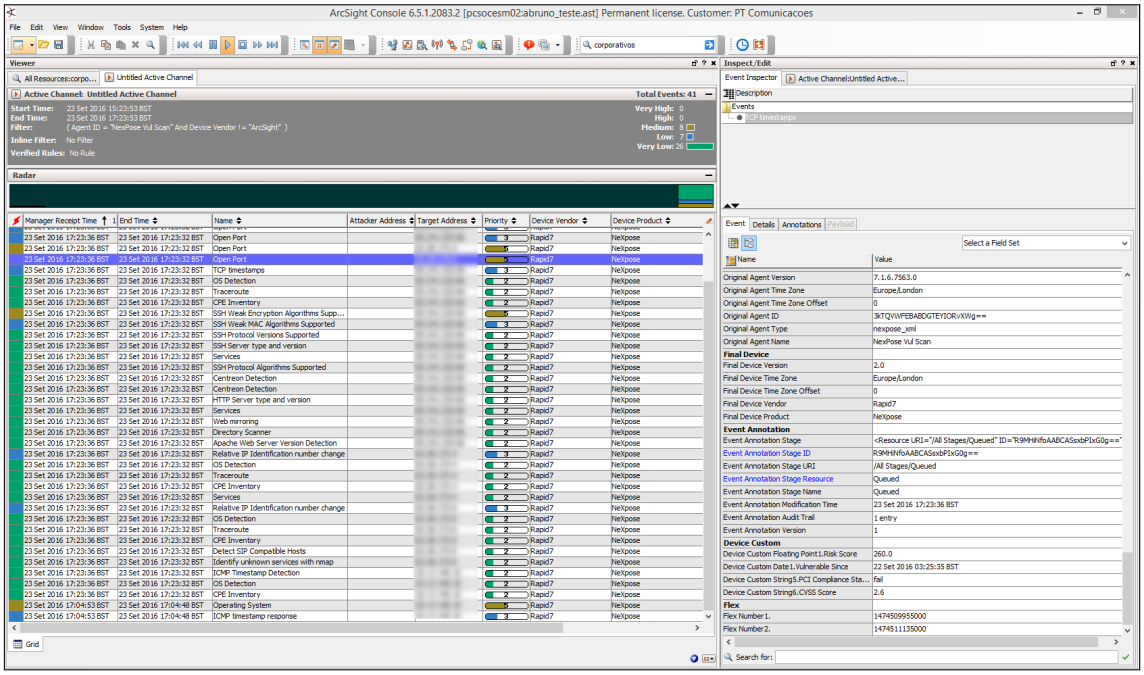

Figura 5.9: Resultados do *scan* do teste 1 no *ArcSight*

5.2 Teste 2 - Janela de disponibilidade de *scan*

A funcionalidade de limitar o tempo de *scan* a uma janela de disponibilidade permitida, para um conjunto de ativos, é demonstrada neste teste. Foi configurado um *scan* espontâneo com as mesmas configurações e ativos que no *scan* do primeiro teste, mas a janela de disponibilidade foi alterada para permitir *scans* das 00h:00m às 17h:56m.

```
2016-09-21 17:43:19 +0100 -- DEBUG -> VAC: SpontaneousScan-TestList2-20160921T174309884 running
2016-09-21 17:43:30 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-21 17:44:15 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-21 17:44:19 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-21 17:57:38 +0100 -- INFO -> VAC: Retrieving SpontaneousScan-TestList2-20160921T174309884 results till time out
2016-09-21 17:57:40 +0100 -- DEBUG -> ScannerManager:
Scan Statistics: SpontaneousScan-TestList2-20160921T174309884
Scan running duration: 802.293329732 seconds
Scan started at : 2016-09-21T17:43:14+01:00
Scan ended at : 2016-09-21T17:56:37+01:00
...
2016-09-21 17:58:15 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-21 17:58:31 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-21 17:58:31 +0100 -- INFO -> ResultsManager:
OpenVAS Scan: openvas-SpontaneousScan-TestList2-20160921T174309884 Sent: 4
```

Figura 5.10: *Log* do teste 2

Como se pode observar pelos excertos do *log* do VAC (Fig. 5.10) o *scan* foi lançado às 17h:43m:15s, realizou a sua operação durante alguns minutos e foi interrompido às 17h:57m:38s. Os resultados obtidos até ao momento da interrupção foram enviados (Fig. 5.11).

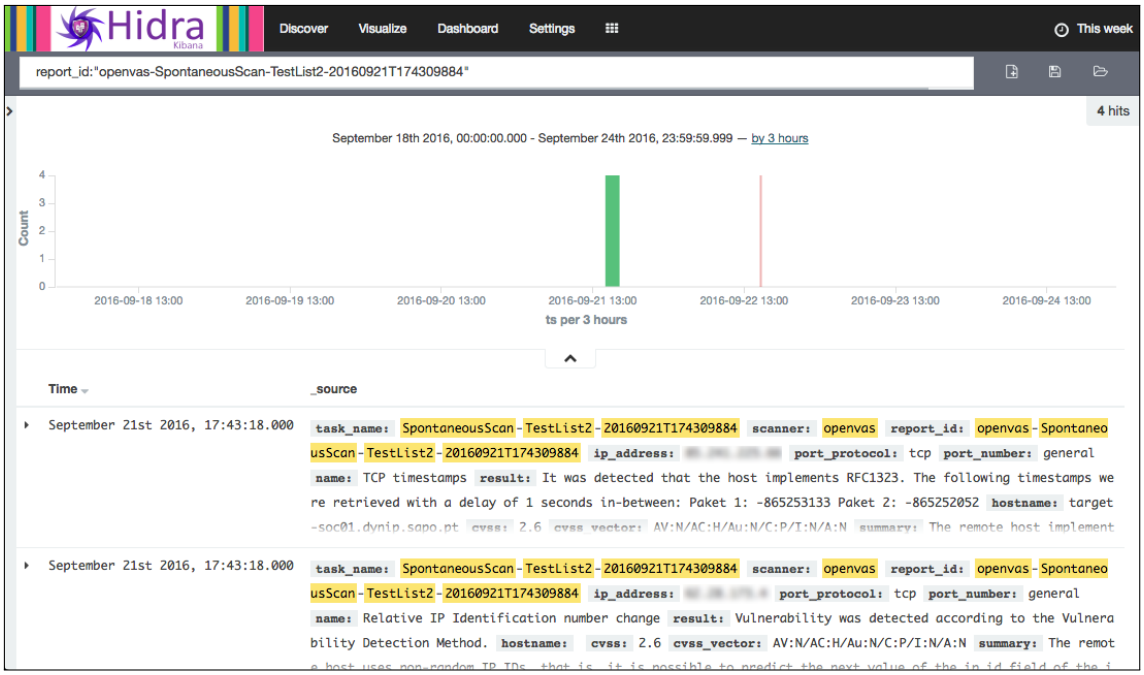

Figura 5.11: Resultados do *scan* do teste 2 no *Kibana*

5.3 Teste 3 - Agendamento de *scan* com frequência diária

Neste teste foi feito o agendamento de um *scan* para um pouco depois da hora em que foi enviado para a aplicação com o objetivo de testar se este era executado à hora exata. Foi escolhida uma frequência diária para perceber se o *scan* era executado no dia seguinte à mesma hora. O *scan* foi configurado com um ativo, o *template* escolhido foi o '*Refined*' e foi configurado para executar todos os dias a partir das 16h15m.

```
2016-09-28 16:15:30 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas2 ACTIVE: 0
2016-09-28 16:15:30 +0100 -- DEBUG -> ScannerManager: SCAN : PeriodicScan-TestList3-20160928T161530477, SCANNER: openvas2
2016-09-28 16:15:32 +0100 -- INFO -> ScannerManager: PeriodicScan-TestList3-20160928T161530477 starting
2016-09-28 16:15:52 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-28 16:15:52 +0100 -- DEBUG -> VAC: PeriodicScan-TestList3-20160928T161530477 running
```

Figura 5.12: *Log* do teste 3 - *scan* primeiro dia

```
2016-09-29 16:15:32 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas2 ACTIVE: 0
2016-09-29 16:15:32 +0100 -- DEBUG -> ScannerManager: SCAN : PeriodicScan-TestList3-20160929T161532844, SCANNER: openvas2
2016-09-29 16:15:39 +0100 -- INFO -> ScannerManager: PeriodicScan-TestList3-20160929T161532844 starting
2016-09-29 16:15:57 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-29 16:15:57 +0100 -- DEBUG -> VAC: PeriodicScan-TestList3-20160929T161532844 running
```

Figura 5.13: *Log* do teste 3 - *scan* segundo dia

Como se pode perceber pelos excertos de *log* da aplicação (Fig 5.12 e 5.13), o *scan* foi executado uma vez por dia depois das 16h:15m. Este teste teve a duração de dois dias. Também podemos ver que os resultados foram integrados no *Elasticsearch* (Fig. 5.14 e 5.15).

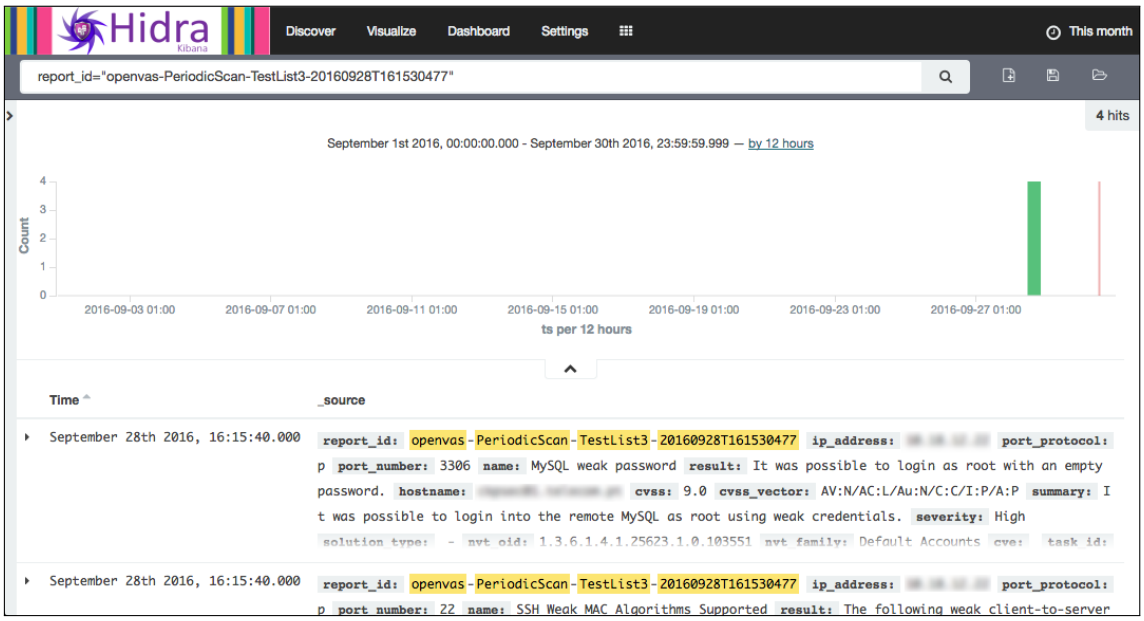

Figura 5.14: Resultados do *scan* do primeiro dia no *Kibana*

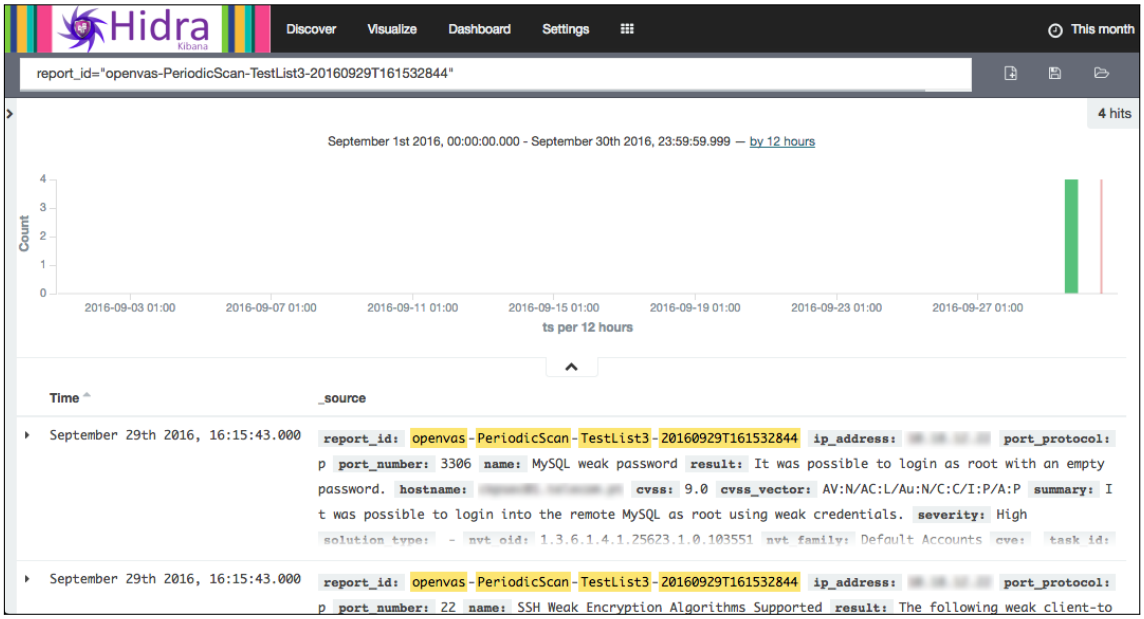

Figura 5.15: Resultados do *scan* do segundo dia no *Kibana*

5.4 Teste 4 - Distribuição de carga entre *vulnerability scanners*

Este teste consistiu em testar o mecanismo de distribuição de carga. Foram lançados dois *scans* com dois *vulnerability scanners OpenVAS* configurados na aplicação.

Através do excerto do *log* do VAC (Fig. 5.16) conseguimos perceber que foram lançados dois *scans* '*SpontaneousScan-TestList4-1-20160922T014115300*' e '*SpontaneousScan-*

```
2016-09-22 01:40:45 +0100 -- INFO -> ScanServer: Request( id: TestList4-1, num_hosts: 1, template: Refined, scanner_type: openvas)
2016-09-22 01:41:15 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-22 01:41:15 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas1 ACTIVE: 0
2016-09-22 01:41:15 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas2 ACTIVE: 0
2016-09-22 01:41:15 +0100 -- DEBUG -> ScannerManager: SCAN : SpontaneousScan-TestList4-1-20160922T014115300, SCANNER: openvas1
2016-09-22 01:41:20 +0100 -- INFO -> ScannerManager: SpontaneousScan-TestList4-1-20160922T014115300 starting
2016-09-22 01:41:22 +0100 -- INFO -> ScanServer: Request( id: TestList4-2, num_hosts: 1, template: Refined, scanner_type: openvas)
2016-09-22 01:41:31 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-22 01:41:40 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-22 01:41:40 +0100 -- DEBUG -> VAC: SpontaneousScan-TestList4-1-20160922T014115300 running
2016-09-22 01:42:20 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-22 01:42:20 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas1 ACTIVE: 1
2016-09-22 01:42:20 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas2 ACTIVE: 0
2016-09-22 01:42:20 +0100 -- DEBUG -> ScannerManager: SCAN : SpontaneousScan-TestList4-2-20160922T014220384, SCANNER: openvas2
2016-09-22 01:42:23 +0100 -- INFO -> ScannerManager: SpontaneousScan-TestList4-2-20160922T014220384 starting
2016-09-22 01:42:31 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-22 01:42:40 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-22 01:42:40 +0100 -- DEBUG -> VAC: SpontaneousScan-TestList4-1-20160922T014115300 running
2016-09-22 01:42:40 +0100 -- DEBUG -> VAC: SpontaneousScan-TestList4-2-20160922T014220384 running
```

Figura 5.16: *Log* do teste 4 - distribuição de carga

TestList4-2-20160922T014220384'. No inicio não havia nenhum *scan* a correr pelo que o *scan* '*SpontaneousScan-TestList4-1-20160922T014115300*' ficou a correr no *scanner* com o identificador '*openvas1*'. Quando o outro *scan* foi iniciado já havia um *scan* a correr no *scanner* '*openvas1*'. Como o scanner '*openvas2*' estava com zero *scans* ativos, esse foi escolhido para executar o novo *scan*.

Nas figuras 5.17 e 5.18 podemos ver os *scans* a correr em instâncias *OpenVAS* diferentes configuradas no VAC.

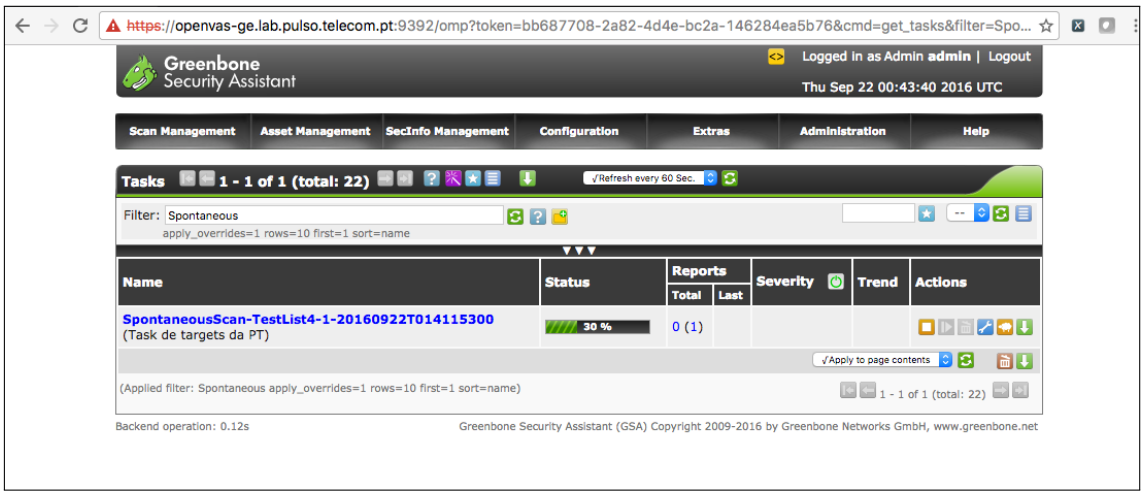

Figura 5.17: Primeiro *scan* a correr na instância *openvas1*

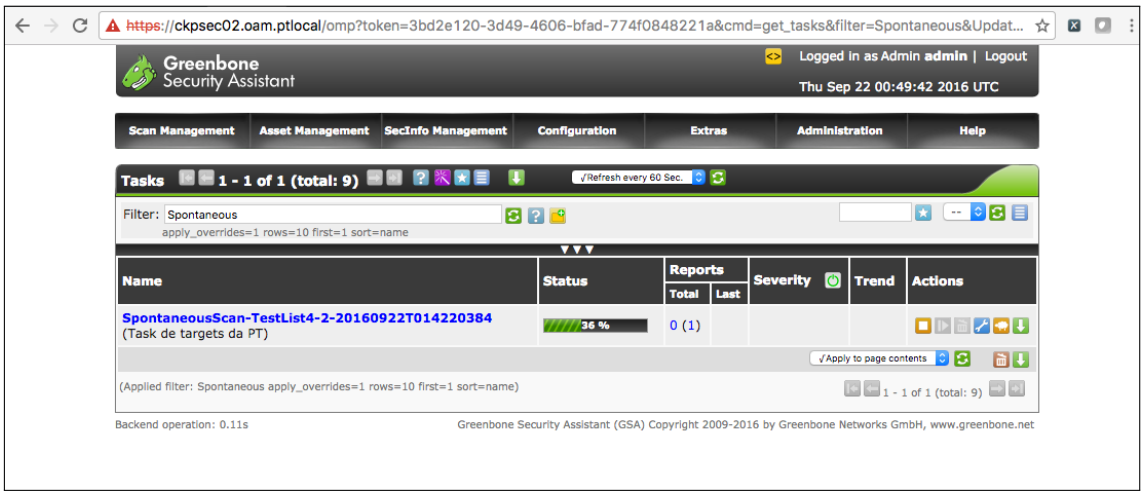

Figura 5.18: Segundo *scan* a correr na instância *openvas2*

5.5 Teste 5 - Configuração de nova tecnologia de *vulnerability scanning*

O carregamento de uma nova tecnologia de *vulnerability scanning* é testada. Para este teste são configuradas na aplicação VAC as bibliotecas previamente criadas para interagir com *vulnerability scanners* da tecnologia *Nexpose* (ficheiros e bibliotecas *Ruby*).

O estado da aplicação foi manipulado para ter apenas configurados *scanners* (Fig. 5.19) e processador de resultados (Fig. 5.22) da tecnologia *OpenVAS*.

A primeira fase consiste na configuração de um *scanner* dessa tecnologia na interface *Web* (Fig. 5.20). O campo '*Scanner Type*' tem que ter o mesmo nome do ficheiro que contém a biblioteca de interação com o *scanner* (neste caso 'nexpose.rb').

Como se pode ver na figura 5.21 a biblioteca foi configurada corretamente.

Depois foi adicionado o processador de dados da tecnologia (Fig. 5.24). O campo '*Processor name*' (Fig. 5.23) também tem que ter o mesmo nome da biblioteca exceto o sufixo '*_processor*' ('nexpose_processor.rb').

Depois foi lançado um *scan* espontâneo com a tecnologia *Nexpose* (Fig. 5.25).

Como visto na figura 5.26 o *scan* foi lançado no *vulnerability scanner Nexpose* e os dados foram enviados para o *Elasticsearch* como visto na interface *Kibana* na figura 5.27.

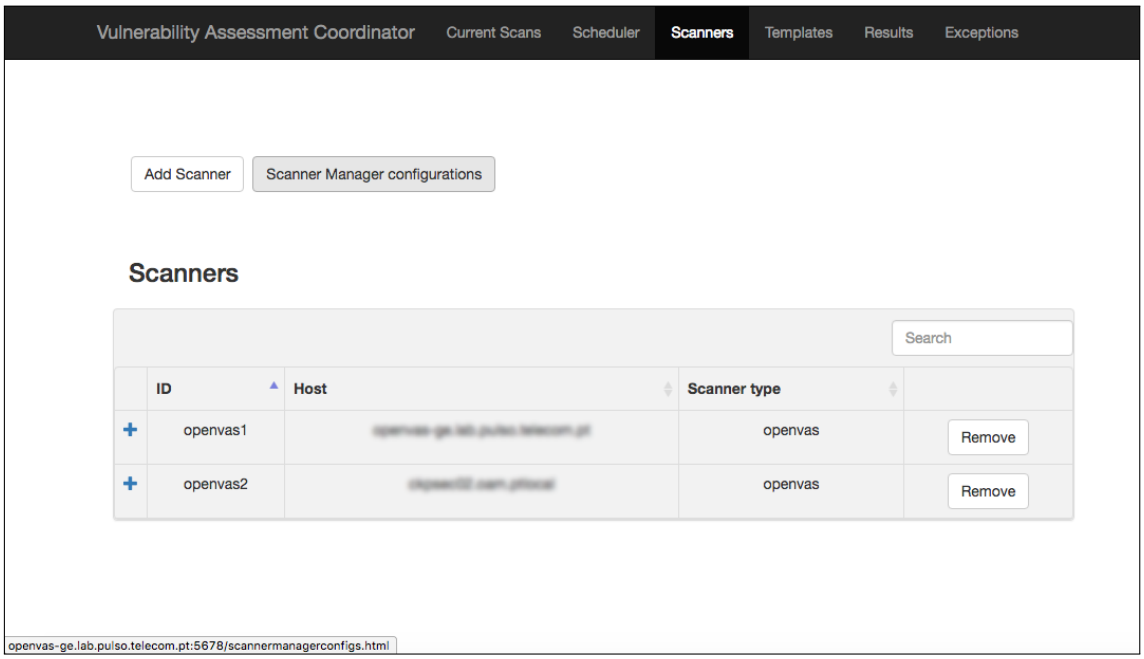

Figura 5.19: *Scanners* instalados no momento com tecnologia *OpenVAS*

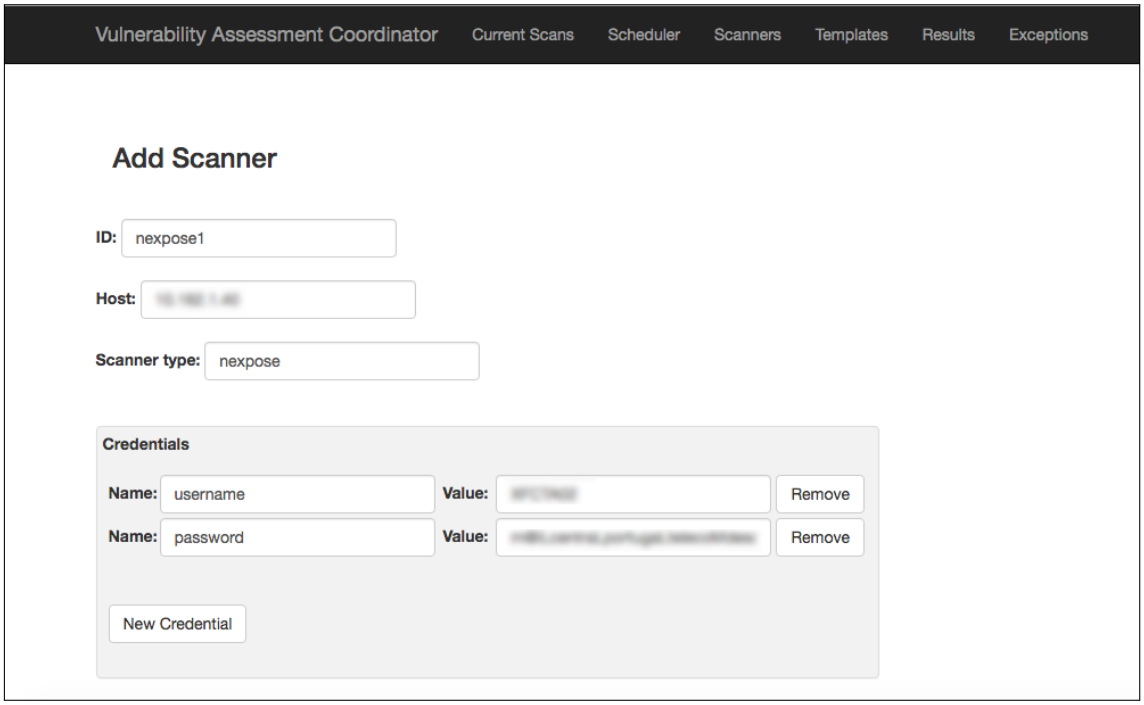

Figura 5.20: Configuração do *scanner Nexpose*

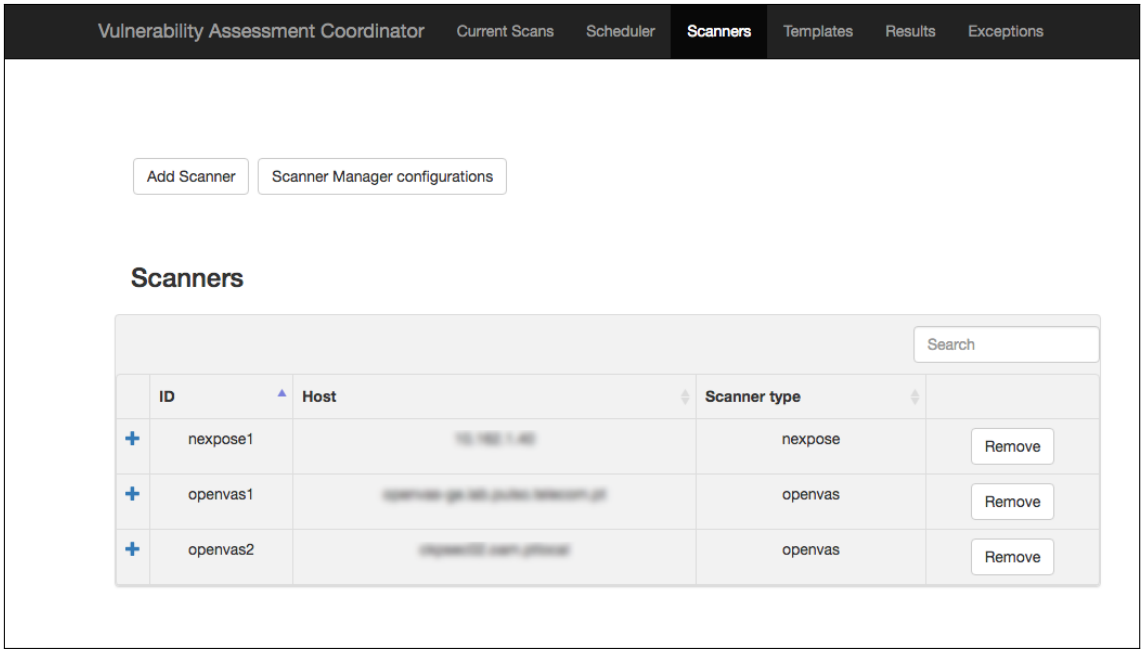

Figura 5.21: *Scanner* Nexpose configurado na aplicação

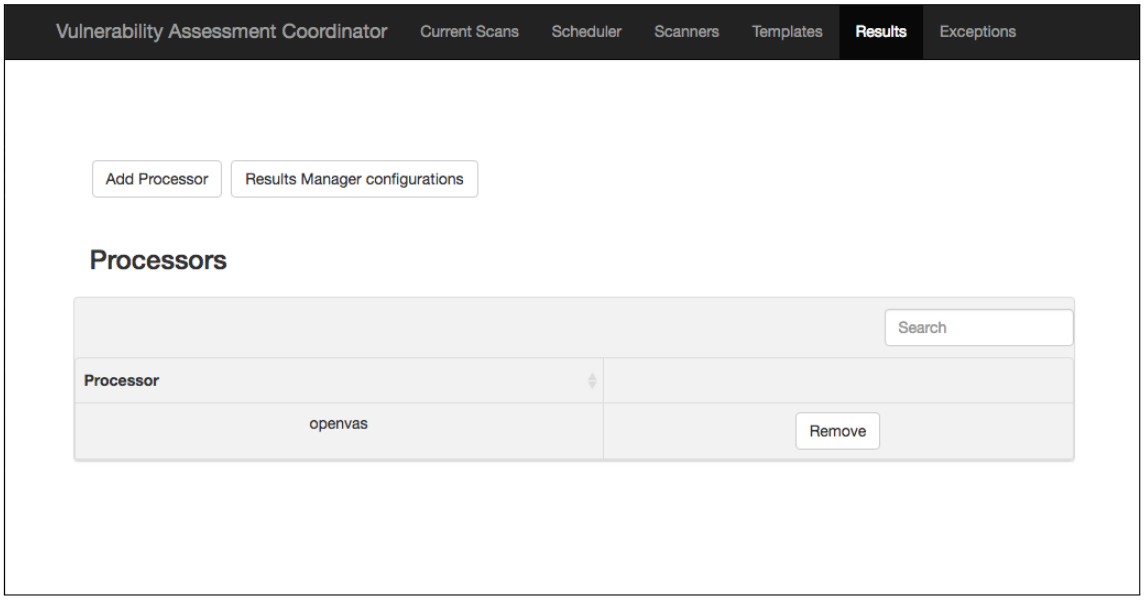

Figura 5.22: Apenas o processador *OpenVAS* instalado na aplicação

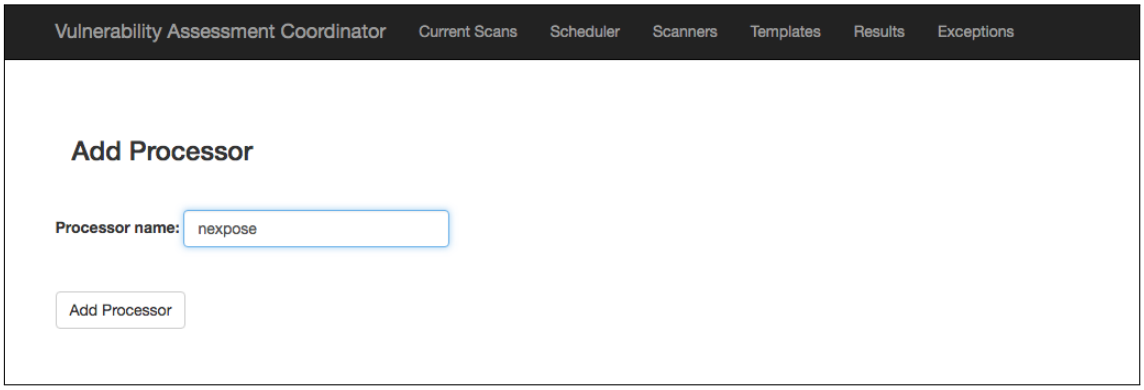

Figura 5.23: Configuração do processador *Nexpose* na aplicação

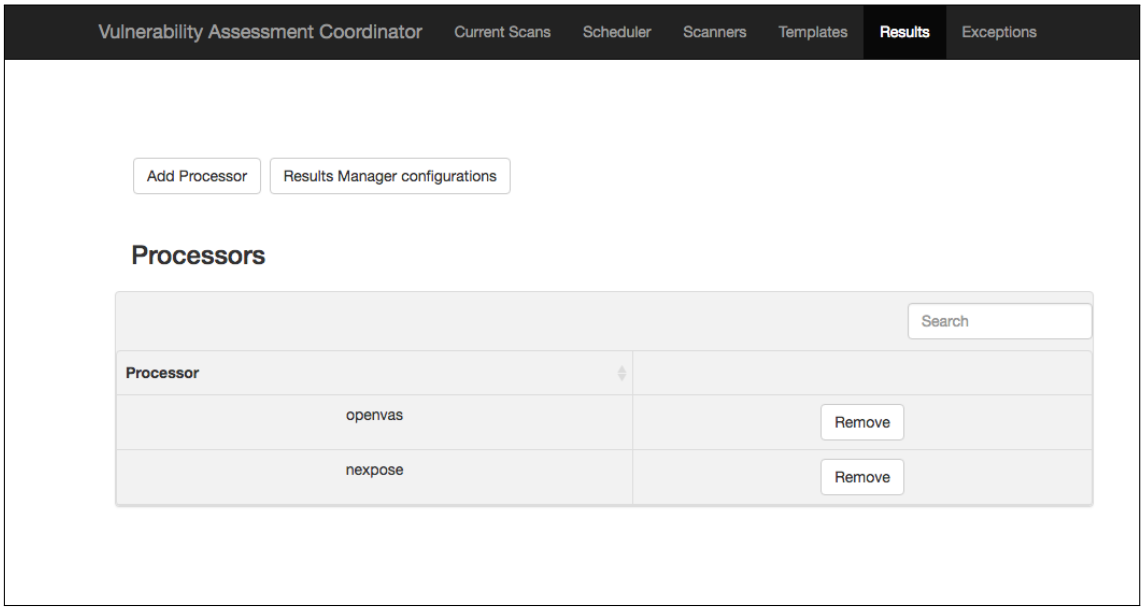

Figura 5.24: Processador *Nexpose* configurado na aplicação

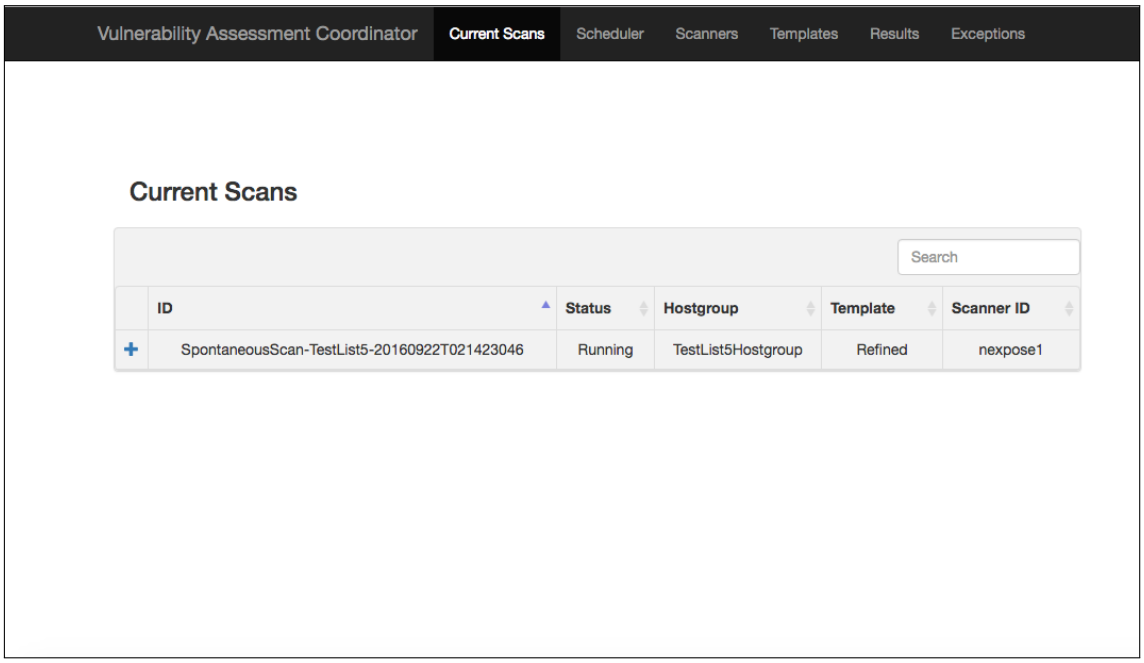

Figura 5.25: *Scan* com tecnologia *Nexpose* a correr na aplicação

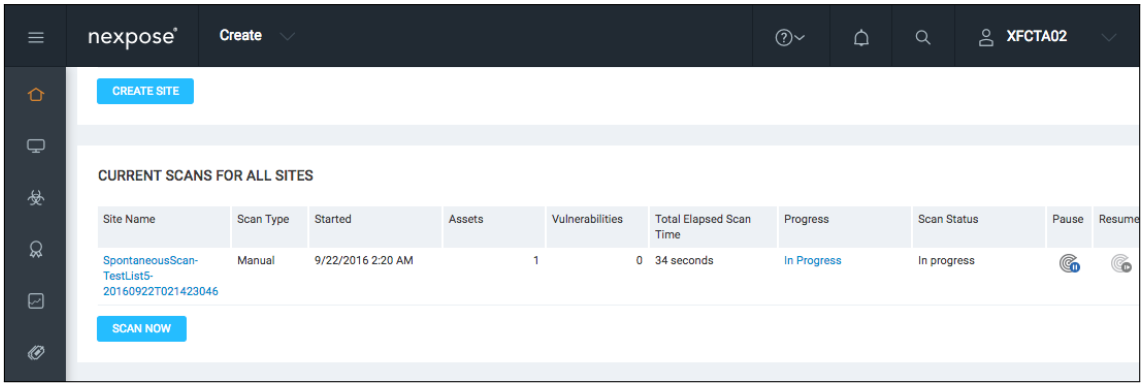

Figura 5.26: Imagem do *scan* a correr no *scanner Nexpose*

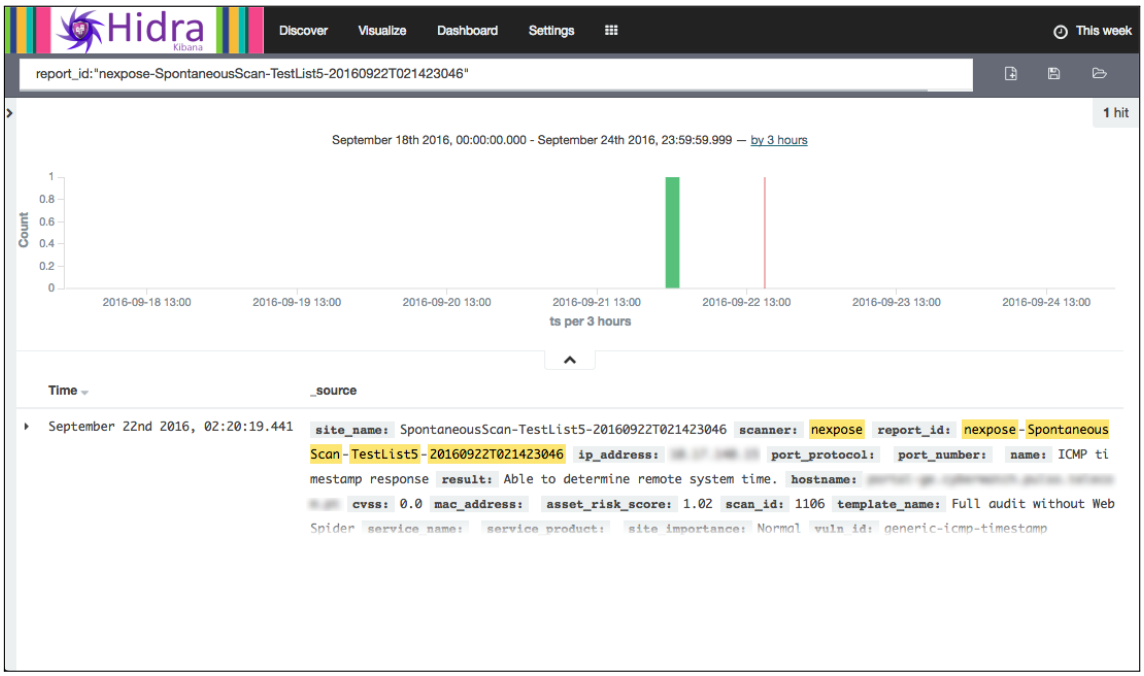

Figura 5.27: Resultados do *scan* integrados no *Kibana*

5.6 Teste 6 - *Scan* de segunda opinião (*Nexpose*)

A funcionalidade de *scan* de segunda opinião é demonstrada neste teste. Foi agendado um *scan* periódico a um ativo com a propriedade '*criticality*' com o valor 9. A janela de disponibilidade para o dia inteiro e o *template* de configuração utilizado foi o '*Refined*'. O *scan* está configurado para ser executado apenas uma vez.

Como se pode ver pelo excerto de *log* (Fig. 5.28) o *scan* foi lançado com a tecnologia de *vulnerability scanning* primária configurada (*OpenVAS*).

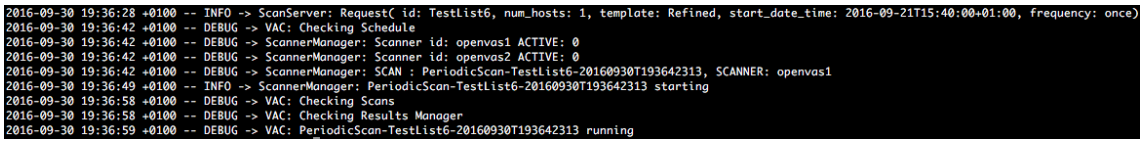

Figura 5.28: *Log* do teste 6 - *scan* inicial

O *scan* termina e, como o nível crítico do ativo é maior ou igual a 9, é lançado um *scan* de segunda opinião com a tecnologia *Nexpose*, como se pode ver pelo *log* na figura 5.29.

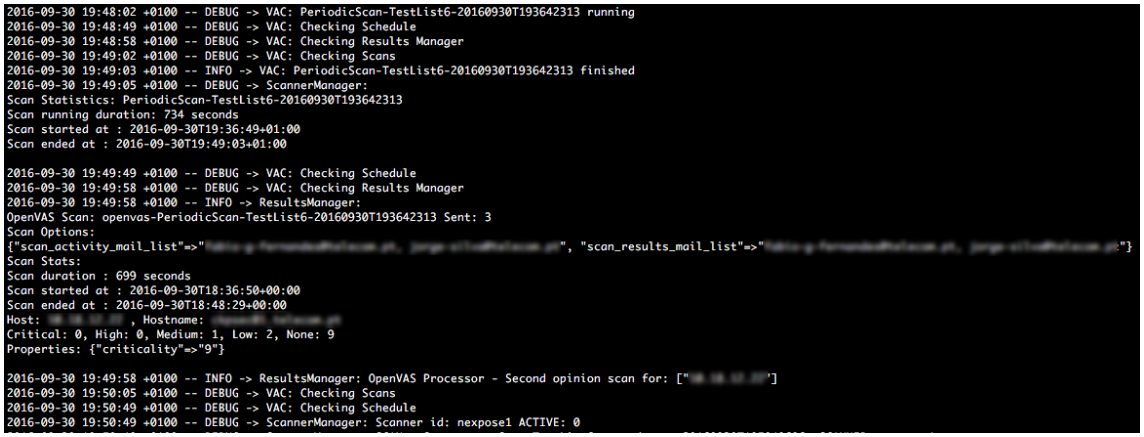

Figura 5.29: *Log* do teste 6 - *scan* de segunda opinião

De seguida podemos observar que os resultados do *scan* inicial com a tecnologia *OpenVAS* e o *scan* de segunda opinião com a tecnologia *Nexpose*, sendo os resultados destes incorporados no *Elasticsearch* (Fig. 5.30 e 5.31).

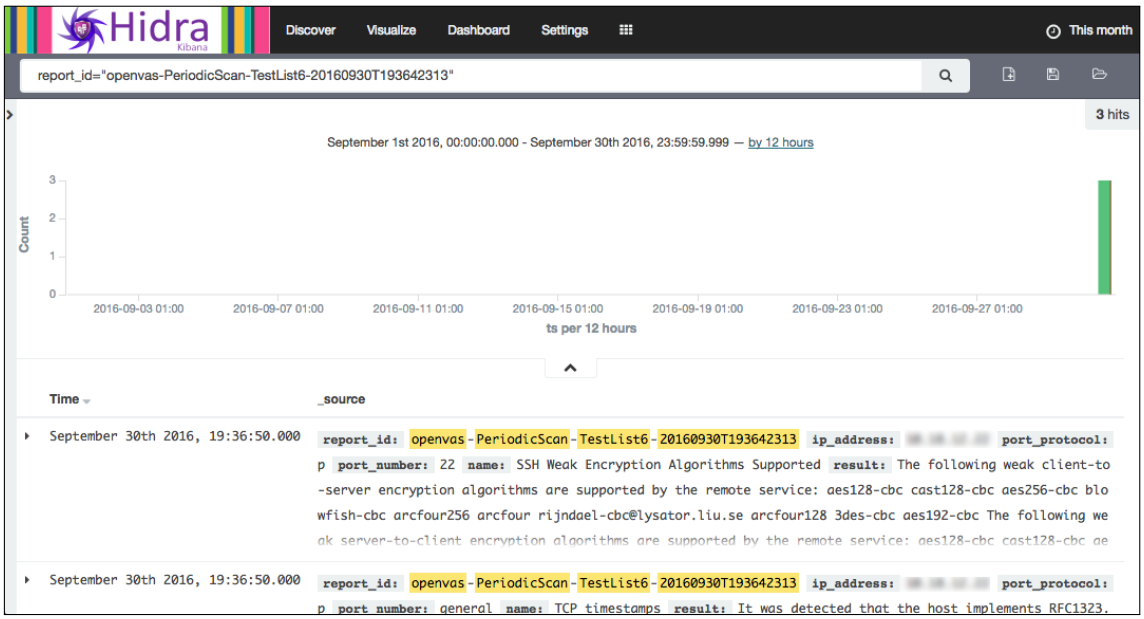

Figura 5.30: Resultados do primeiro*scan*

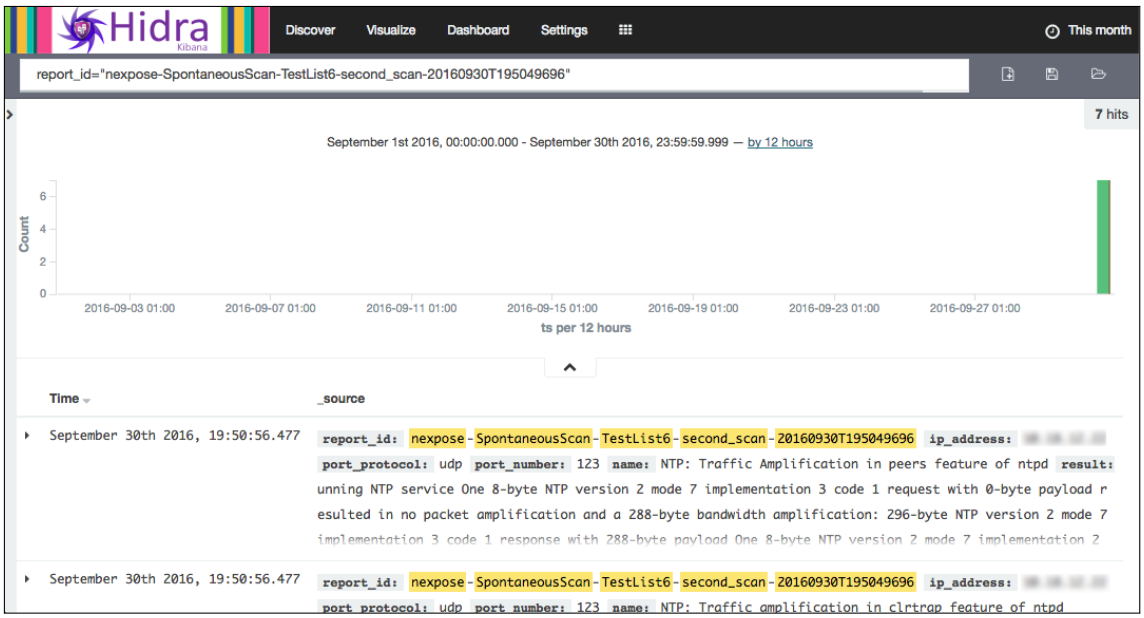

Figura 5.31: Resultados do *scan* de segunda opinião

5.7 Teste 7 - Escalabilidade da aplicação

A escalabilidade da aplicação foi testada através da observação do desempenho de *scans* a vários ativos com vários *scanners*. Foram medidos os tempos de execução dos *vulnerability scanners* e da aplicação. Todos os *scans* foram feitos com a tecnologia *OpenVAS*, pois sendo a tecnologia primária do VAC vai ser a mais utilizada.

5.7.1 Teste com um e dois *scanners* a dois ativos

	Scanner			VAC		
	Scan 1	*Scan* 2	Total	*Scan* 1	*Scan* 2	Total
1 *Scanner*	759 s	927 s	927 s	780 s	954 s	954 s
2 *Scanners*	665 s	488 s	665 s	718 s	553 s	718 s

Tabela 5.1: Resultados do teste com um e dois *scanners* a dois ativos

Os resultados apresentados na tabela 5.1 mostram que com a utilização de dois *scanners*, a aplicação apresenta uma melhoria de 24% aproximadamente.

5.7.2 Teste com um e dois *scanners* a quatro ativos

	Scanner				
	Scan 1	Scan 2	Scan 3	Scan 4	Total
1 Scanner	934 s	1293 s	1527 s	1593 s	1593 s
2 Scanner	638	515 s	1072 s	649 s	1072 s
	VAC				
	Scan 1	Scan 2	Scan 3	Scan 4	Total
1 Scanner	1011 s	1331 s	1562 s	1612 s	1612 s
2 Scanners	675 s	543 s	1096 s	660 s	1096 s

Tabela 5.2: Resultados do teste com um e dois *scanners* a quatro ativos

Com quatro ativos o teste demonstra (Tabela 5.2) que com dois *scanners* conseguimos obter uma melhoria de 32% aproximadamente.

Este teste demonstra a vantagem da escalabilidade da aplicação VAC, mas não podemos esquecer que nem sempre pode ocorrer uma melhoria significativa devido ao facto de estes *scans* serem bastante influenciados pelo ambiente de operação onde são realizados, como referido na secção 3.4.

5.8 Teste 8 - Mecanismo de filtragem de resultados falso positivos

A funcionalidade de filtragem de resultados falso positivos é comprovada. Inicialmente foi feito um *scan* ao ativo teste onde foram encontradas cinco vulnerabilidades:

- *SSH Weak MAC Algorithms Supported.*
- *SSH Weak Encryption Algorithms Supported.*
- *SNMP GETBULK Reflected DrDoS.*

- *Report default community names of the SNMP Agent.*
- *TCP timestamps.*

Como se pode ver pela interface *Kibana* na figura 5.32, nos resultados do *scan* foram encontradas cinco vulnerabilidades.

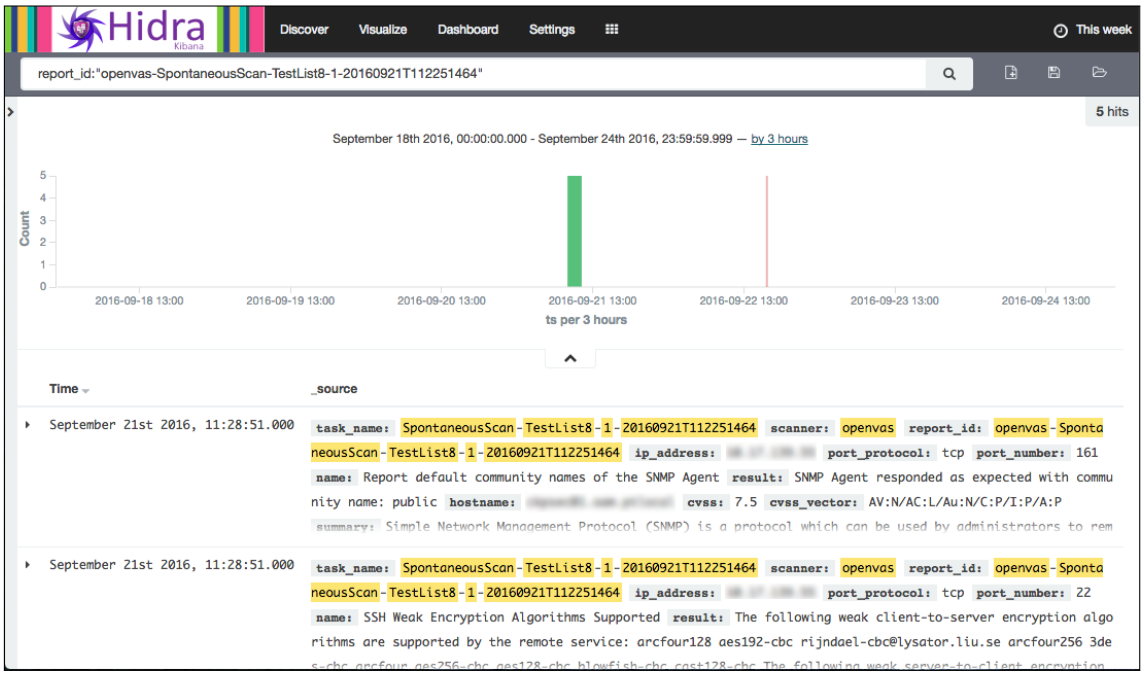

Figura 5.32: Resultados do *scan* inicial

Para testar a funcionalidade foi adicionada à lista de exceções:

- *Name*: *Report default community names of the SNMP Agent.*
- *Host*: 10.17.139.55 .
- *Port Protocol*: tcp.
- *Port Number*: 161.
- *Result*: *SNMP Agent responded as expected with community name: public.*
- *Scanner Type: openvas.*
- *Reason*: Razão teste.
- *Person*: Fábio Fernandes.
- *Contact*: mail@teste.pt .

Após a configuração da exceção foi lançado o mesmo *scan* com as mesmas configurações e foram encontradas as seguintes vulnerabilidades:

- *SSH Weak MAC Algorithms Supported.*
- *SSH Weak Encryption Algorithms Supported.*
- *SNMP GETBULK Reflected DrDoS.*
- *TCP timestamps.*

Foram apenas encontradas quatro vulnerabilidades (Fig. 5.33) desaparecendo aquela que foi excepcionada (*Report default community names of the SNMP Agent*).

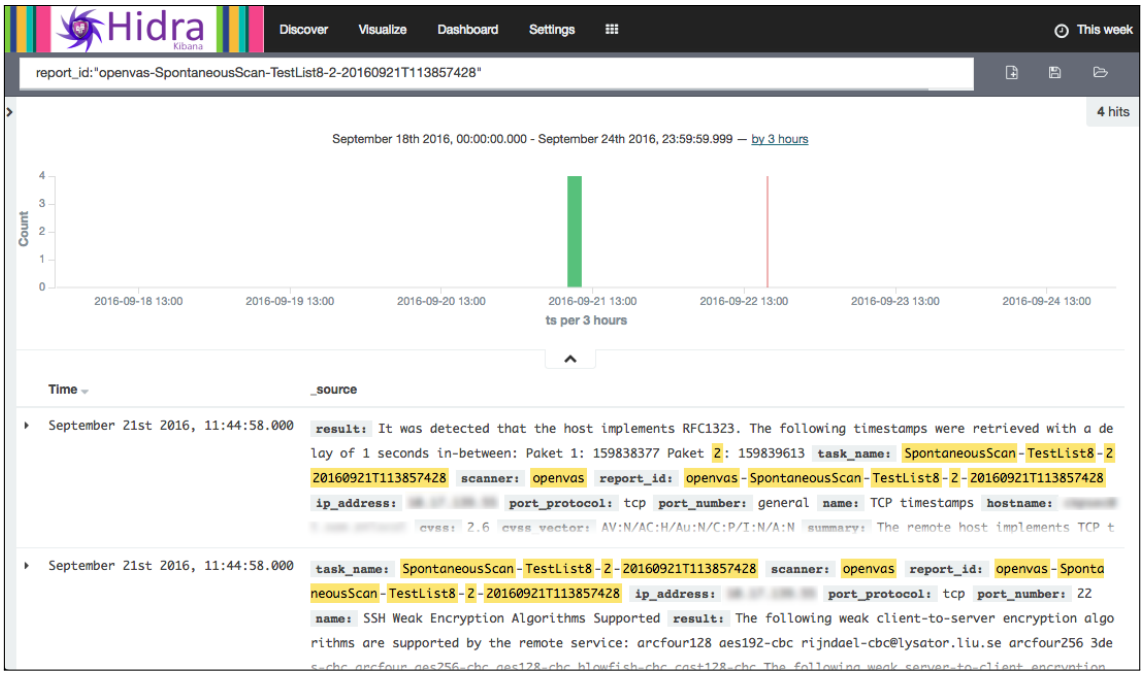

Figura 5.33: Resultados do *scan* após a adição de um falso positivo

5.9 Teste 9 - Acompanhamento da evolução de um ativo

Este teste avalia a capacidade da aplicação permitir acompanhar a evolução da alteração do número de vulnerabilidades presentes nos ativos. Para esse efeito foi configurado um ativo controlado e, ao longo dos vários *scans*, foram feitas e desfeitas configurações vulneráveis para verificar se estas apareciam nos resultados.

Foi feito um *scan* inicial onde foram encontradas três vulnerabilidades (Fig. 5.34).

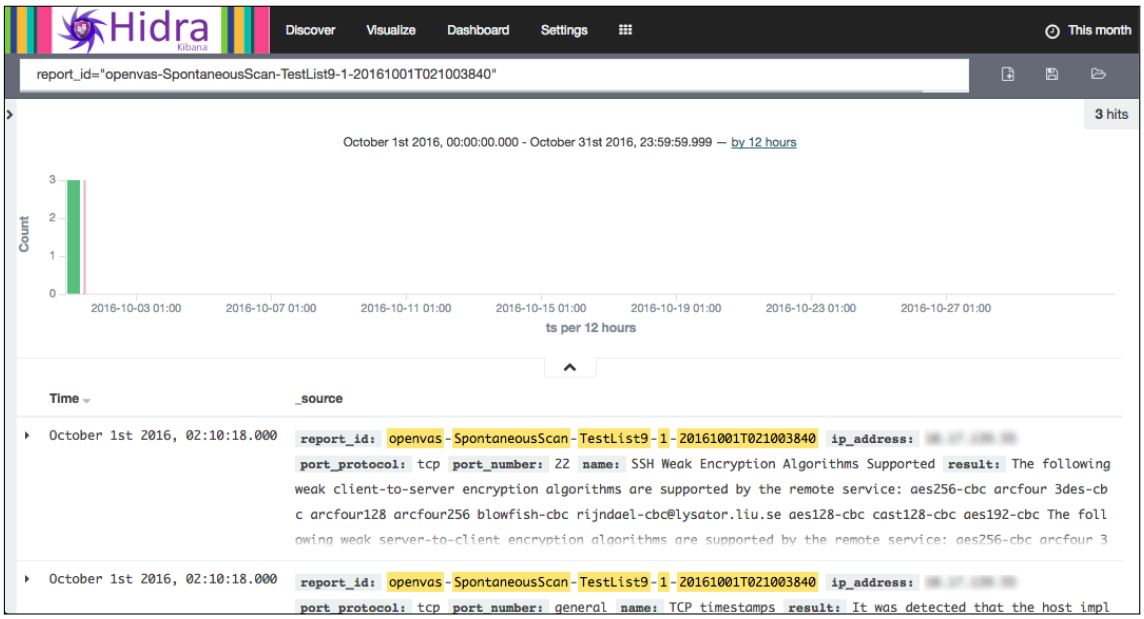

Figura 5.34: Evolução de um ativo: *scan* inicial

De seguida foram ligados os serviços *MySQL*, *PostgreSQL* e *Postfix* com configurações vulneráveis e realizado um *scan* que descobriu três vulnerabilidades adicionais associadas ao serviços expostos (Fig 5.35).

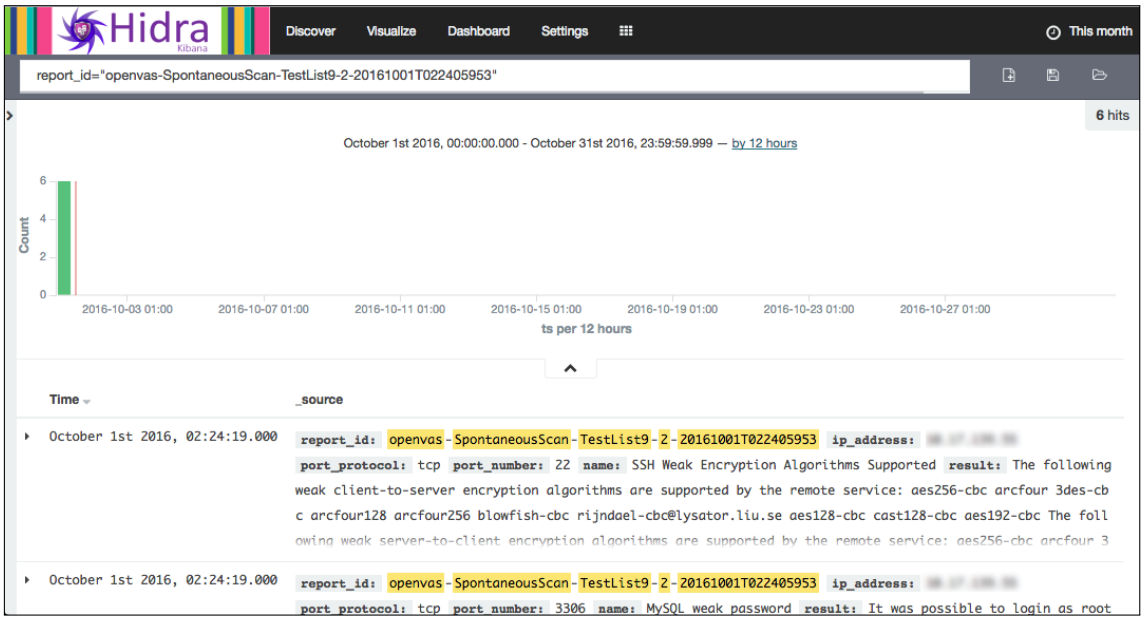

Figura 5.35: Evolução de um ativo: aumento de vulnerabilidades descobertas

Foram desativados os três serviços com configurações vulneráveis e foram feitos dois *scans* que apresentam ambos as três vulnerabilidades iniciais (Fig. 5.36 e 5.37).

Iniciou-se o serviço *SNMP* com uma configuração vulnerável e foi lançado um *scan* que descobriu 2 vulnerabilidades associadas a esse serviço (Fig. 5.38).

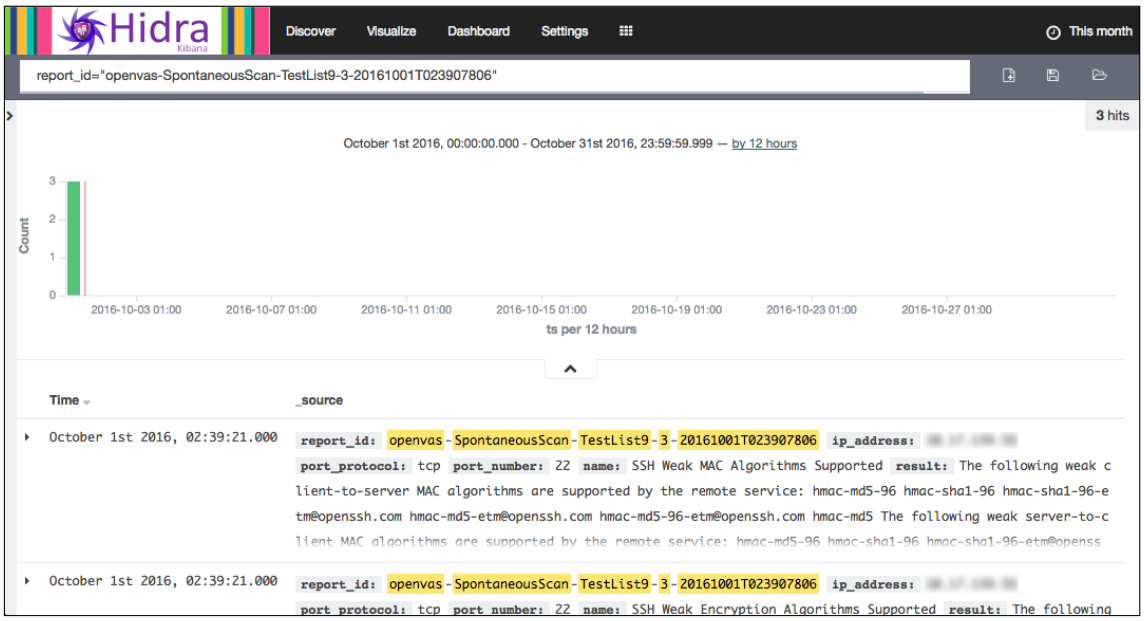

Figura 5.36: Evolução de um ativo: correção de vulnerabilidades *scan* 1

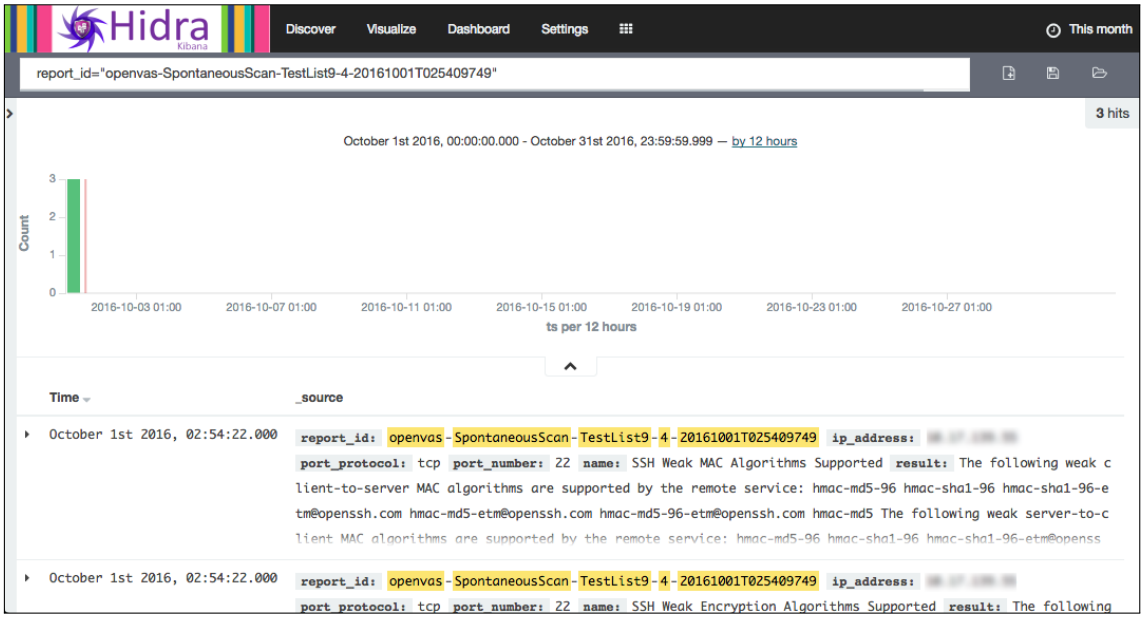

Figura 5.37: Evolução de um ativo: correção de vulnerabilidades *scan* 2

No final foi desligado o serviço *SNMP* com as configurações vulneráveis e foram descobertas três vulnerabilidades novamente (Fig 5.39).

A figura 5.40 apresenta um gráfico, criado na interface *Kibana*, da evolução do número de vulnerabilidades ao longo dos *scans* realizados neste teste provando que é possível acompanhar a evolução das vulnerabilidades apresentadas por um ativo ao longo do tempo.

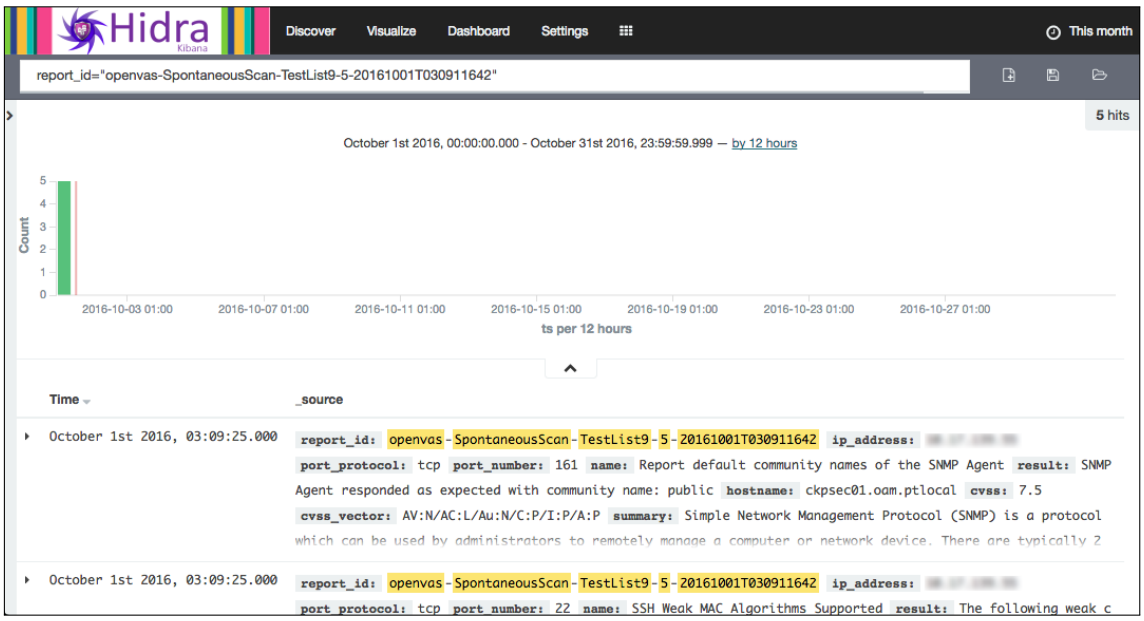

Figura 5.38: Evolução de um ativo: vulnerabilidades SNMP

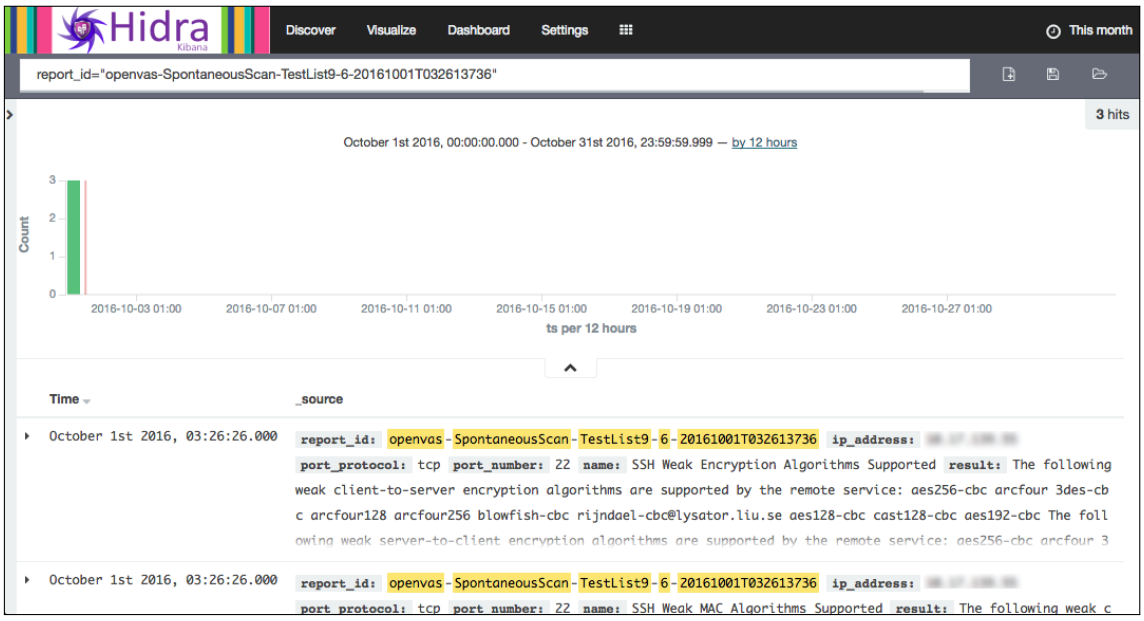

Figura 5.39: Evolução de um ativo: *scan* final

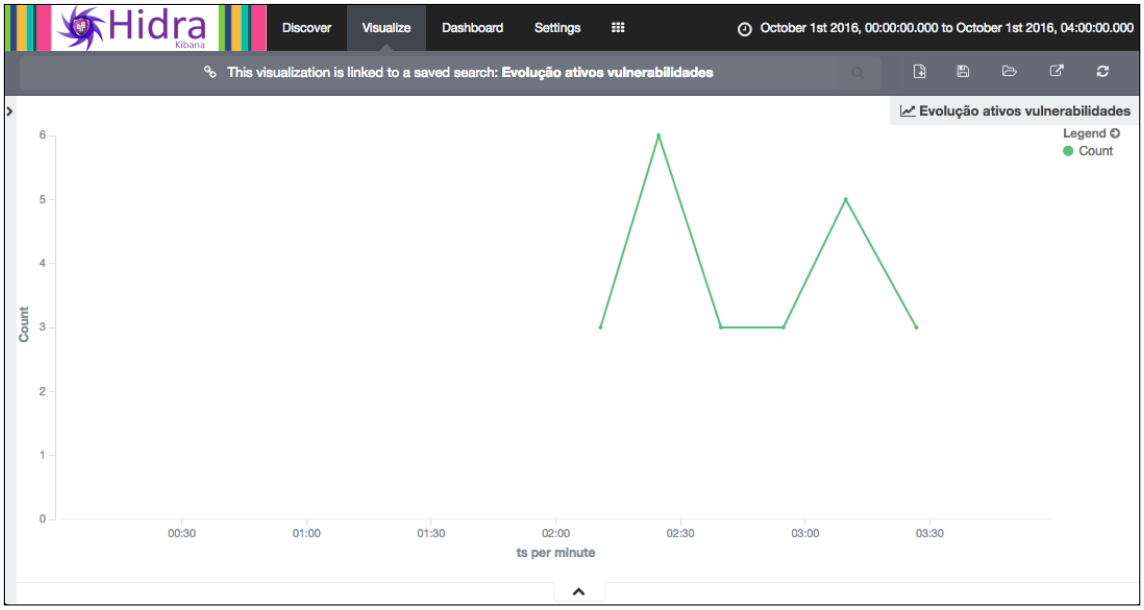

Figura 5.40: Evolução de um ativo: gráfico no Kibana

5.10 Teste 10 - Afinação de *template* de configuração *OpenVAS*

O *OpenVAS*, por omissão, tem uma configuração muito abrangente, o que pode tornar os *scans* mais precisos mas prejudica gravemente o seu desempenho. Foram feitas afinações que permitiram encontrar um ponto ótimo entre precisão e desempenho como explicado na secção 4.1. Essas afinações resultaram no *template* de configuração '*Refined*' que consiste na melhoria da eficiência do teste a portos TCP, na redução drástica do número de portos testados e na desativação dos *plugins* de testes a aplicações *Web*.

Foram realizados dois *scans* ao mesmo ativo, um testa todos os portos TCP e outro testa um número de portos reduzidos resultantes da afinação feita. Os resultados estão presentes na tabela 5.3 e mostram que houve uma redução aproximada de 77% no tempo de execução, no entanto não foram negligenciados os testes aos portos de serviços importantes pelo que esta afinação representa um bom compromisso entre precisão e desempenho.

	Scanner	VAC
Todos os portos TCP	3062 s	3123 s
Configuração afinada	673 s	718 s

Tabela 5.3: Resultados de *scan OpenVAS* com e sem afinação

5.11 Teste 11 - Falha de um *vulnerability scanner*

A continuação de um *scan* no caso de falha e recuperação de um *vulnerability scanner* é testada. Foi lançado um *scan* sobre um ativo e passado algum tempo do *scan* ter iniciado foi simulada uma falha no *vulnerability scanner*.

```
2016-09-27 10:41:37 +0100 -- DEBUG -> VAC: PeriodicScan-TestList11-20160927T103535846 running
2016-09-27 10:41:38 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-27 10:42:35 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-27 10:42:37 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-27 10:42:38 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-27 10:42:44 +0100 -- ERROR -> ScannerManager: OpenVAS openvas2 - Exception while trying to check if scan id 04c26806-8476-4133-a5f5-6c06d491bf71 is finished
2016-09-27 10:42:44 +0100 -- ERROR -> ScannerManager: OpenVAS openvas2 - Exception while trying to check if scan id 04c26806-8476-4133-a5f5-6c06d491bf71 is running
2016-09-27 10:42:44 +0100 -- ERROR -> ScannerManager: OpenVAS openvas2 - Exception while trying to check if scan id 04c26806-8476-4133-a5f5-6c06d491bf71 is finished
2016-09-27 10:42:44 +0100 -- ERROR -> ScannerManager: OpenVAS openvas2 - Exception while trying to check if scan id 04c26806-8476-4133-a5f5-6c06d491bf71 is paused
2016-09-27 10:42:44 +0100 -- ERROR -> ScannerManager: OpenVAS openvas2 - Exception while trying to check if scan id 04c26806-8476-4133-a5f5-6c06d491bf71 is running
2016-09-27 10:43:35 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-27 10:43:38 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-27 10:43:44 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-27 10:43:46 +0100 -- ERROR -> ScannerManager: OpenVAS openvas2 - Exception while trying to start OpenVAS scan id: PeriodicScan-TestList11-20160927T103535846
2016-09-27 10:43:47 +0100 -- INFO -> VAC: PeriodicScan-TestList11-20160927T103535846 not started
2016-09-27 10:44:35 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-27 10:44:38 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-27 10:44:47 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-27 10:44:48 +0100 -- INFO -> VAC: PeriodicScan-TestList11-20160927T103535846 starting 2
```

Figura 5.41: *Log* do teste 11

Como se pode observar no *log* (Fig. 5.41) a aplicação deixou de conseguir perceber o estado do *scan* e depois recuperou a sua execução mais tarde, terminando ao enviar os resultados para o Hidra, como se pode observar na figura 5.42.

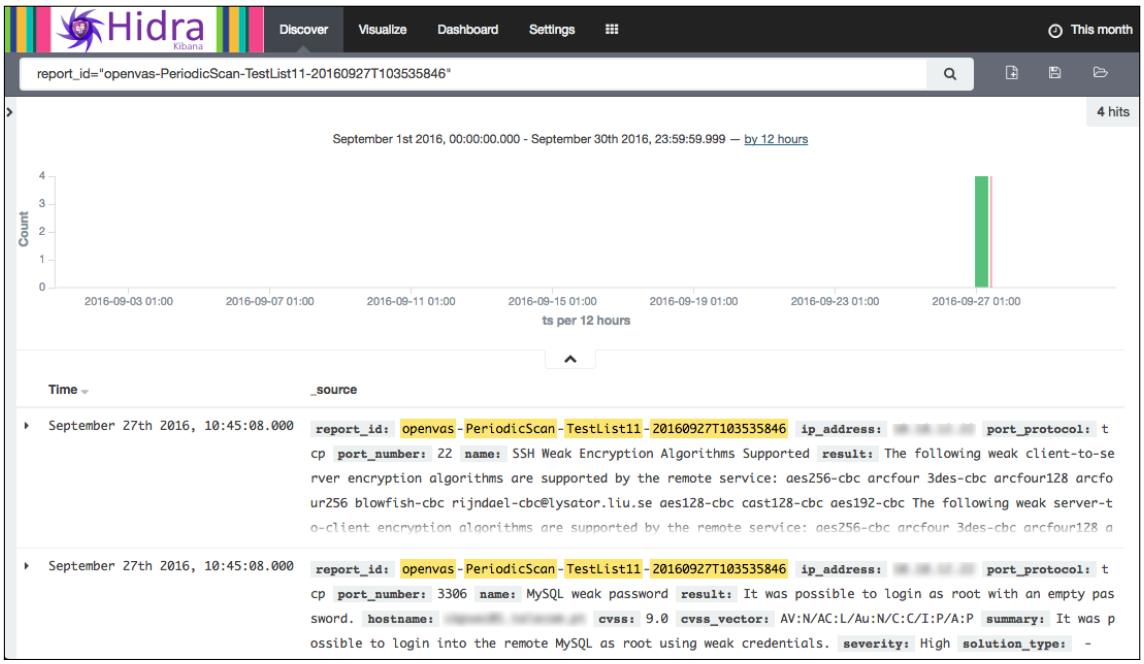

Figura 5.42: Resultados do *scan*

5.12 Teste 12 - Tolerância a falhas da aplicação

Neste teste foi demonstrada a capacidade da aplicação preservar o seu estado na ocorrência de uma falha da aplicação.

```
2016-09-27 12:29:37 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas2 ACTIVE: 0
2016-09-27 12:29:37 +0100 -- DEBUG -> ScannerManager: SCAN : PeriodicScan-TestList12-20160927T122936630, SCANNER: openvas2
2016-09-27 12:29:43 +0100 -- INFO -> ScannerManager: PeriodicScan-TestList12-20160927T122936630 starting
2016-09-27 12:30:36 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-27 12:30:36 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-27 12:30:36 +0100 -- DEBUG -> VAC: PeriodicScan-TestList12-20160927T122936630 running
2016-09-27 12:30:43 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-27 12:31:36 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-27 12:31:36 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-27 12:31:37 +0100 -- DEBUG -> VAC: PeriodicScan-TestList12-20160927T122936630 running
2016-09-27 12:31:43 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-27 12:35:39 +0100 -- INFO -> VAC: Building Scanner Manager
2016-09-27 12:35:40 +0100 -- INFO -> VAC: Building Scheduler
2016-09-27 12:35:40 +0100 -- INFO -> VAC: Building Results Manager
2016-09-27 12:35:41 +0100 -- DEBUG -> VAC: Checking Schedule
2016-09-27 12:35:41 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-09-27 12:35:41 +0100 -- DEBUG -> VAC: Checking Scans
2016-09-27 12:35:41 +0100 -- DEBUG -> VAC: PeriodicScan-TestList12-20160927T122936630 running
2016-09-27 12:36:41 +0100 -- DEBUG -> VAC: Checking Schedule
```

Figura 5.43: *Log* do teste 12

Como se pode observar no *log* (Fig. 5.43) foi lançado um *scan* e passado algum tempo, enquanto este estava em execução, foi interrompida a aplicação, aproximadamente às 12h:31m. Passados alguns minutos foi lançada a aplicação que reconstruiu o seu estado e retomou o acompanhamento do *scan* que estava previamente em execução. No fim o *scan* terminou e os dados foram enviados para o Hidra (Fig. 5.44).

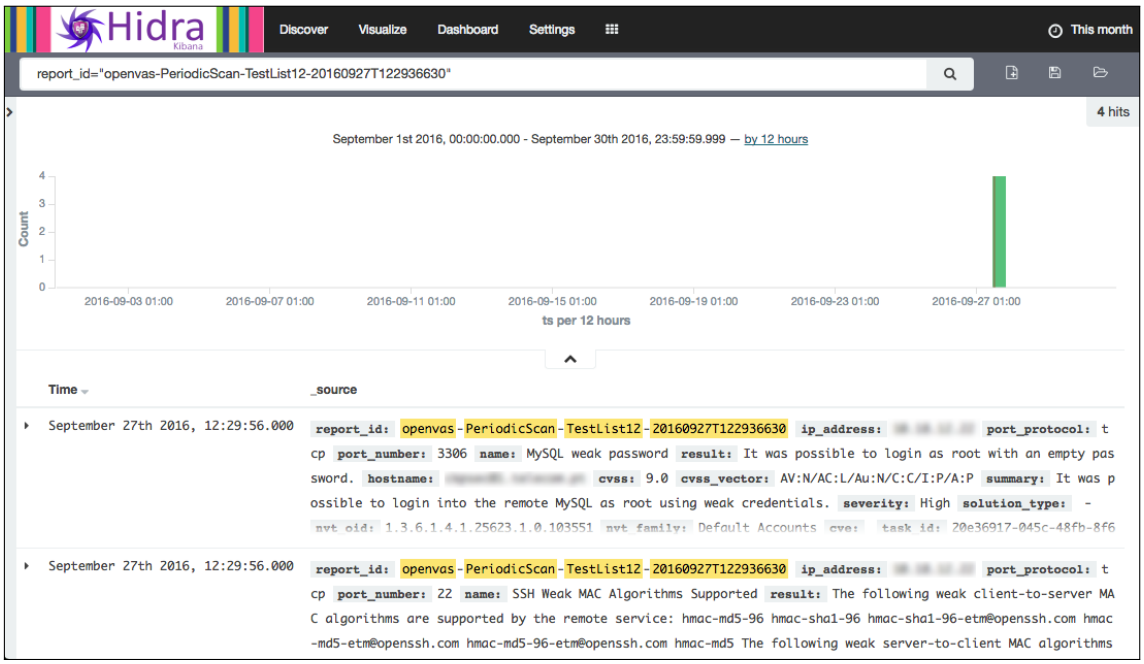

Figura 5.44: Resultados do *scan* no Hidra

5.13 Teste 13 - Falha de um ativo alvo

No teste final é verificado que a aplicação suporta a falha de um ativo alvo de *scan*. Foi lançado um *scan* sobre um ativo e durante a execução desse *scan* foi desligado o ativo com o comando '*shutdown -P 0*'.

```
2016-10-04 09:51:56 +0100 -- INFO -> ScanServer: Request( id: TestList13, num_hosts: 1, template: Refined, scanner_type: openvas)
2016-10-04 09:52:01 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-10-04 09:52:05 +0100 -- DEBUG -> VAC: Checking Scans
2016-10-04 09:52:41 +0100 -- DEBUG -> VAC: Checking Schedule
2016-10-04 09:52:41 +0100 -- DEBUG -> ScannerManager: Scanner id: openvas2 ACTIVE: 0
2016-10-04 09:52:41 +0100 -- DEBUG -> ScannerManager: SCAN : SpontaneousScan-TestList13-20161004T095241879, SCANNER: openvas2
2016-10-04 09:52:48 +0100 -- INFO -> ScannerManager: SpontaneousScan-TestList13-20161004T095241879 starting
2016-10-04 09:53:01 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-10-04 09:53:05 +0100 -- DEBUG -> VAC: Checking Scans
2016-10-04 09:53:05 +0100 -- DEBUG -> VAC: SpontaneousScan-TestList13-20161004T095241879 running
...
2016-10-04 09:57:06 +0100 -- DEBUG -> VAC: SpontaneousScan-TestList13-20161004T095241879 running
2016-10-04 09:57:48 +0100 -- DEBUG -> VAC: Checking Schedule
2016-10-04 09:58:01 +0100 -- DEBUG -> VAC: Checking Results Manager
2016-10-04 09:58:06 +0100 -- DEBUG -> VAC: Checking Scans
2016-10-04 09:58:07 +0100 -- INFO -> VAC: SpontaneousScan-TestList13-20161004T095241879 finished
2016-10-04 09:58:08 +0100 -- DEBUG -> ScannerManager:
Scan Statistics: SpontaneousScan-TestList13-20161004T095241879
Scan running duration: 319 seconds
Scan started at : 2016-10-04T09:52:48+01:00
Scan ended at : 2016-10-04T09:58:07+01:00
```

Figura 5.45: *Log* do teste 13

Como se pode observar pelo *log* da aplicação na figura 5.45, o *vulnerability scanner* detetou o ativo como estando em baixo e terminou o *scan*. A aplicação detetou que o *scan* terminou, retirou os resultados intermédios e enviou-os para o Hidra (Fig. 5.46).

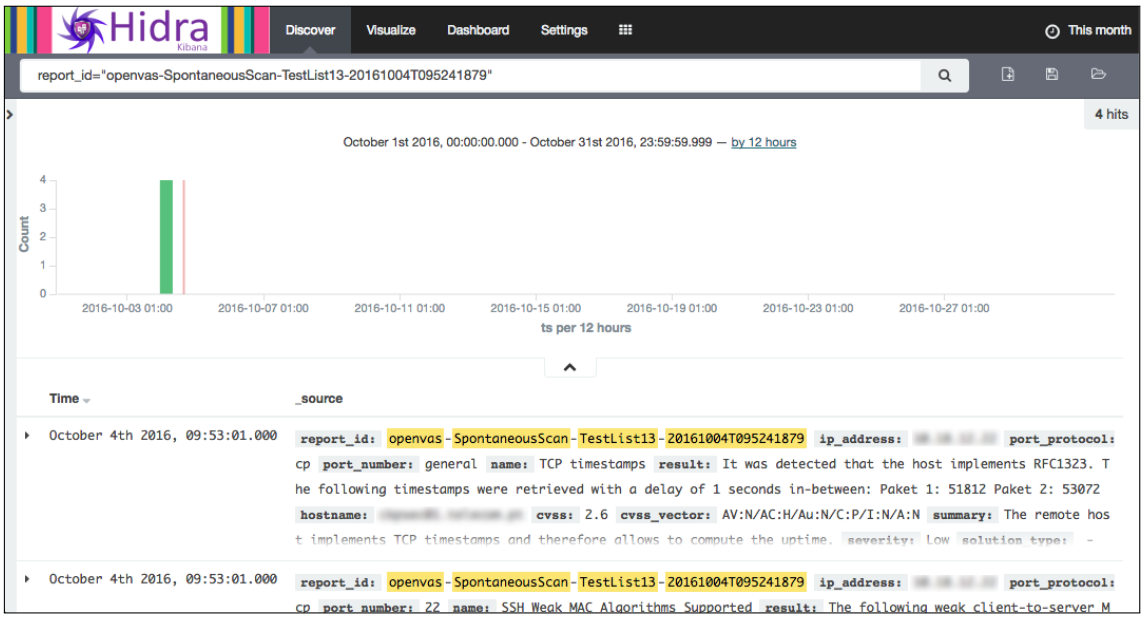

Figura 5.46: Resultados do *scan* no Hidra

Capítulo 6

Conclusão e Trabalho Futuro

A aplicação VAC desenvolvida neste projeto focou-se nos requisitos propostos inicialmente (secção 3.2) e, como demonstrado no capítulo da Avaliação (cap. 5), estes foram cumpridos. Através da possibilidade da configuração de *scans* periódicos e a integração automática dos resultados nas plataformas de armazenamento, a aplicação permite que *scans* sejam lançados sobre ativos de forma automática possibilitando uma avaliação contínua da sua segurança em termos de vulnerabilidades. Adicionalmente, a afinação das configurações de *vulnerability scanners*, em especial o *OpenVAS*, permite que sejam obtidos resultados em tempo útil, operação essencial num ambiente composto por um grande número de ativos.

A automatização concretizada pela aplicação permite aliviar a equipa que faz a gestão de vulnerabilidades da tarefa repetitiva de gerir uma grande quantidade de *scans* em vários *vulnerability scanners*, extrair os resultados e integra-los nas plataformas. Isto possibilita que o tempo desta equipa possa ser gasto noutras tarefas de maior valor, como, por exemplo, o acompanhamento da resolução das vulnerabilidades nos ativos com as equipas responsáveis. Adicionalmente, a funcionalidade de *scans* espontâneos permite responder rapidamente a situações de emergência, como, por exemplo, a divulgação de uma vulnerabilidade crítica num serviço comum. A possibilidade de janelas de disponibilidade por dia nos dois tipos de *scan* permite minimizar o impacto da qualidade de serviço dos ativos.

A interação com a aplicação é feita de forma fácil através da interface *Web* disponibilizada. A aplicação pode acompanhar a evolução das necessidades da DCY em termos de tecnologias de *vulnerability scanning* e plataformas de armazenamento de resultados, pois permite adicionar módulos que representam estas na aplicação. A capacidade da integração de vários *vulnerability scanners* da mesma tecnologia e a distribuição dos *scans* por estas, tendo em conta a sua carga na criação do *scan*, garante que a solução é escalável, sendo este um requisito importante no ambiente de grande escala onde a aplicação vai atuar. A criação de um mecanismo simples de adição e filtragem de resultados falso positivos permite melhorar a precisão dos

resultados enviados para as plataformas de armazenamento e, consequentemente, melhora a eficácia das equipas que utilizam estes. Como em qualquer projeto existem melhorias a fazer e, com o limite de tempo e contexto académico deste, foi feito um maior investimento na automatização, eficiência e eficácia dos *scans*, ou seja, no componente gestor de *scanners* e no componente agendamento.

Consequentemente são expostos alguns pontos a melhorar na concretização do VAC. O armazenamento de falso positivos foi concretizado de uma forma simples através de uma *Hash* que guarda falso positivos em memória. Como é óbvio este mecanismo é pouco escalável pelo que deve ser utilizada uma base de dados que seja eficiente, com prioridade na leitura para suportar um grande número de consultas (uma por cada resultado de um *scan*).

A definição de grupos de ativos e suas configurações são feitas no agendamento de um *scan*, caso se queira criar outro agendamento com o mesmo grupo tem que se repetir a configuração, pelo que deve ser criado um registo de grupos de ativos para facilitar a reutilização destas configurações.

Apesar de no componente de integração a adição de tecnologias ser modular, a adição de novas plataformas de armazenamento obriga à alteração dos processadores de dados de todas as tecnologias. Para tornar este componente mais flexível deve-se aumentar a separação entre os processos de leitura e de transformação dos dados.

É essencial também criar mecanismos de autenticação no acesso à ferramenta, para evitar acesso à informação sensível que esta contem e para evitar que um atacante possa utilizar a aplicação como uma ferramenta de disrupção da rede e serviços que esta mesma ajuda a proteger. Para concretizar isto é preciso criar mecanismos de autenticação na API *Web*, comunicação segura entre os clientes e a aplicação (HTTPS), e cifra do estado guardado em disco pela aplicação.

Como demonstrado, a informação que a aplicação envia para o Hidra permite fazer o acompanhamento do estado de segurança em termos de vulnerabilidade dos ativos alvos de *scan*. Com vista a aliviar o trabalho da equipa que faz a gestão de vulnerabilidades na DCY é importante criar um mecanismo automático de criação de relatórios que em cada *scan* realizado faça um relatório e o envie para a equipa técnica responsável pela configuração das máquinas presentes no *scan*.

Outro ponto importante será o calculo de KPIs (*Key Performance Indicator*) que permitam, num dado instante no tempo, avaliar o estado da segurança dos ativos na empresa, para permitir à equipa de gestão perceber em que áreas é mais prioritário investir na configuração segura de máquinas.

Abreviaturas

ACK Acknowledgement.

AD Active Directory.

AJAX Asynchronous Javascript and XML.

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

ARP Address Resolution Protocol.

AVI Ataque Vulnerabilidade Intrusão.

CIDR Classless Inter-Domain Routing.

CLI Command-Line Interface.

CNA CVE Numbering Authorities.

CSRF Cross-Site Request Forgery.

CTO Chief Technology Officer.

CVE Common Vulnerabilities and Exposures.

CVSS Common Vulnerability Scoring System.

DCY Direção de Cyber Security e Privacidade.

DDoS Distributed Denial of Service.

DHCP Dynamic Host Configuration Protocol.

DSL Domain-Specific Language.

FTP File Transfer Protocol.

HP Hewlett-Packard.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS HyperText Transfer Protocol Secure.

IANA Internet Assigned Numbers Authority.

ICMP Internet Control Message Protocol.

IDS Intrusion detection System.

IP Internet Protocol.

IPS Intrusion Prevention System.

JSON JavaScript Object Notation.

KB Knowledge Base.

KPI Key Performance Indicator.

LDAP Lightweight Directory Access Protocol.

MAC Message Authentication Code.

NVT Network Vulnerability Test.

OMP OpenVAS Management Protocol.

OTP OpenVAS Transfer Protocol.

PC Personal Computer.

PCI Payment Card Industry.

PDF Portable Document Format.

PT Portugal Telecom.

REST Representational State Transfer.

RTT Round Trip Time.

SIEM Security Information and Event Management.

SNMP Simple Network Management Protocol.

SQL Structured Query Language.

SSH Secure Shell.

SSL Secure Socket Layer.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UDP User Datagram Protoco.

UML Unified Modeling Language.

URL Uniform Resource Locator.

VAC Vulnerability Assessment Coordinator.

WAF Web Application Firewall.

XML Extensible Markup Language.

XSS Cross-Site Scripting.

YAML YAML Ain't Markup Language.

Bibliografia

[1] ArcSight SIEM. https://www.microfocus.com/pt-br/products/siem-security-information-event-management/overview. Accesso em: 2019-06-16.

[2] Benchmarking serialization speed of YAML, JSON, Marshal, and MessagePack in Ruby, JRuby, and Rubinius · GitHub. https://gist.github.com/havenwood/4513627#file-benchmark_results-rb. Accesso em: 2019-03-07.

[3] Black Hat and Defcon 2008 Presentation Video and Slides. https://nmap.org/presentations/BHDC08/. Accesso em: 2019-03-02.

[4] Bootstrap · The most popular HTML, CSS, and JS library in the world. https://getbootstrap.com/. Accesso em: 2019-03-07.

[5] Browsing and Enumeration. http://www.sans.edu/research/security-laboratory/article/attacks-browsing. Accesso em: 2015-12-06.

[6] Common Vulnerability Scoring System v3.0: Specification Document. https://www.first.org/cvss/specification-document. Accesso em: 2015-12-06.

[7] Comodo hacker: I hacked DigiNotar too; other CAs breached. http://arstechnica.com/security/2011/09/comodo-hacker-i-hacked-diginotar-too-other-cas-breached/. Accesso em: 2016-02-03.

[8] CVE – Common Vulnerabilities and Exposures. http://cve.mitre.org/. Accesso em: 2015-12-06.

[9] Elasticsearch - RESTful, Distributed Search and Analytics. https://www.elastic.co/pt/products/elasticsearch. Accesso em: 2019-06-16.

[10] GitHub - msgpack/msgpack-ruby: MessagePack implementation for Ruby. https://github.com/msgpack/msgpack-ruby. Accesso em: 2019-03-07.

[11] GitHub kost/openvas-omp-ruby. https://github.com/kost/openvas-omp-ruby. Accesso em: 2019-03-05.

[12] GitHub rapid7/nexpose-client. https://github.com/rapid7/nexpose-client. Accesso em: 2019-03-05.

[13] Greenbone Security Manager – User Manual. http://www.openvas.org/documentation.html. Accesso em: 2015-12-06.

[14] Heartbleed Bug. http://heartbleed.com/. Accesso em: 2015-12-06.

[15] IDFAQ: Which backdoors live on which ports? https://www.2600index.info/Links/33/3/www.sans.org/security-resources/idfaq/which-backdoors-live-on-which-ports/8/4.html. Accesso em: 2019-03-02.

[16] Implementing a Vulnerability Management Process. https://www.sans.org/reading-room/whitepapers/threats/implementing-vulnerability-management-process-34180. Accesso em: 2015-12-06.

[17] Kibana - Explore, Visualize, Discover Data. https://www.elastic.co/pt/products/kibana. Accesso em: 2019-06-16.

[18] MessagePack: It's like JSON. but fast and small. https://msgpack.org/. Accesso em: 2019-03-07.

[19] Messaging that just works - RabbitMQ. https://www.rabbitmq.com/. Accesso em: 2019-06-16.

[20] Metasploit. http://www.metasploit.com/. Accesso em: 2015-12-06.

[21] Nessus, OpenVAS and Nexpose VS Metasploitable. https://hackertarget.com/nessus-openvas-nexpose-vs-metasploitable/. Accesso em: 2015-12-06.

[22] Nexpose User's Guide. https://community.rapid7.com/docs/DOC-1387. Accesso em: 2015-12-06.

[23] Nmap Internet Research Project. http://research.nmap.org/. Accesso em: 2019-03-02.

[24] Nmap NASL Preferences. https://docs.greenbone.net/GSM-Manual/gos-3.1/en/scan_configuration.html#nmap-nasl-preferences. Accesso em: 2019-03-02.

[25] Nmap Network Scanning - Timing and Performance. https://nmap.org/book/man-performance.html. Accesso em: 2019-02-26.

[26] Nmap: The Network Mapper. https://nmap.org/. Accesso em: 2015-12-06.

[27] Offensive Security Exploit Database Archive. https://www.exploit-db.com/. Accesso em: 2015-12-06.

[28] OpenVAS – Open Vulnerability Assessment System. http://www.openvas.org/. Accesso em: 2015-12-06.

[29] OWASP Top Ten Project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Accesso em: 2015-12-06.

[30] Rapid7 – Nexpose. http://www.rapid7.com/products/nexpose/. Accesso em: 2015-12-06.

[31] Ruby XML Performance Shootout: Nokogiri vs LibXML vs Hpricot vs REXML. http://www.rubyinside.com/ruby-xml-performance-benchmarks-1641.html. Accesso em: 2019-03-07.

[32] SecTools.org Top Network Security Tools. http://sectools.org/. Accesso em: 2015-12-06.

[33] Service Name and Transport Protocol Port Number Registry. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. Accesso em: 2019-03-02.

[34] ShellShock: All you need to know about the Bash Bug vulnerability. http://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability. Accesso em: 2015-12-06.

[35] Sinatra. http://sinatrarb.com. Accesso em: 2019-03-07.

[36] The Lifecycle of a Vulnerability. http://www.iss.net/documents/whitepapers/ISS_Vulnerability_Lifecycle_Whitepaper.pdf. Accesso em: 2015-12-06.

[37] VASCO Announces Bankruptcy Filing by DigiNotar B.V. https://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_vasco_announces_bankruptcy_filing_by_diginotar_bv.aspx. Accesso em: 2016-02-03.

[38] Well Known Port List: nmap-services. `https://nmap.org/book/nmap-services.html`. Accesso em: 2019-03-02.

[39] Miguel Correia and Paulo Sousa. *Segurança no Software*. FCA, 1 edition, 2010.

[40] Paulo Veríssimo and Luís Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.

[41] Lawrie Brown William Stallings. *Computer Security Principles and Practice*. Prentice Hall Press, 2 edition, 2012.